The Dissertation Committee for Bryan Connor Silverthorn
certifies that this is the approved version of the following dissertation:

A Probabilistic Architecture for Algorithm Portfolios

Committee:

Risto Miikkulainen, Supervisor

Bart Selman

Peter Stone

Adam Klivans

Pradeep Ravikumar

A Probabilistic Architecture for Algorithm Portfolios

by

Bryan Connor Silverthorn, B.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2012

For Sarah, who sees the best in me.

Acknowledgments

This dissertation was reached—inevitably—on a meandering path, but that path was made far smoother by the guidance of my advisor, Risto Miikkulainen. His consistently constructive advice and his unshakeable optimism made this work possible.

Thanks, too, are due to my fellow conspirators in directional statistics, Joe Reisinger and Austin Waters, who gave me something completely different to think about, and a great excuse to cite Machiavelli. I am also immensely grateful for the understanding of Michael Madole and Sydney Jones, two friends who put up with, on a daily basis, my occasionally grudging awareness of the outside world. The equal patience and love shown by my family has been a firm anchor when I have needed it.

Thanks most of all to The Flightpath, which I visited rarely; but often enough.

Finally, this entire research area owes much to the tireless efforts of past and present SAT competition organizers, particularly Olivier Roussel, Daniel Le Berre, and Laurent Simon, as well as the countless solver authors who have made automated logical reasoning the impressive tool that it is, and who have willingly seen their work assimilated into the portfolio collective.

BRYAN CONNOR SILVERTHORN

The University of Texas at Austin

May 2012

A Probabilistic Architecture for Algorithm Portfolios

Publication No. _____

Bryan Connor Silverthorn, Ph.D.
The University of Texas at Austin, 2012

Supervisor: Risto Miikkulainen

Heuristic algorithms for logical reasoning are increasingly successful on computationally difficult problems such as satisfiability, and these *solvers* enable applications from circuit verification to software synthesis. Whether a problem instance can be solved, however, often depends in practice on whether the correct solver was selected and its parameters appropriately set. *Algorithm portfolios* leverage past performance data to automatically select solvers likely to perform well on a given instance. Existing portfolio methods typically select only a single solver for each instance. This dissertation develops and evaluates a more general portfolio method, one that computes complete solver execution schedules, including repeated runs of nondeterministic algorithms, by explicitly incorporating probabilistic reasoning into its operation. This *modular architecture for probabilistic portfolios* (MAPP) includes novel solutions to three issues central to portfolio operation: first, it estimates solver performance distributions from limited data by constructing a generative model; second, it integrates domain-specific information by predicting instances on which solvers exhibit similar performance; and, third, it computes execution schedules using an efficient and effective dynamic programming approximation. In a series of empirical comparisons designed to replicate past solver competitions, MAPP outperforms the most prominent al-

ternative portfolio methods. Its success validates a principled approach to portfolio operation, offers a tool for tackling difficult problems, and opens a path forward in algorithm portfolio design.

Contents

Acknowledgments **v**

Abstract **vi**

Contents **viii**

List of Tables **xii**

List of Figures **xiii**

Acronyms **xv**

Symbols **xviii**

Chapter 1 Introduction **1**

1.1 Motivation 2

1.2 Why Algorithm Portfolios? 3

1.3 General Portfolio Operation 4

1.4 The Modular Architecture for Probabilistic Portfolios 6

1.5 Outline of the Dissertation 9

Chapter 2 Background **11**

2.1 Satisfiability 11

2.1.1 SAT and Its Solvers . . . 12
2.1.2 Solving SAT is Important . . . 15
2.1.3 Solving SAT is Difficult (But Not *Too* Difficult) . . . 17
2.1.4 Solving SAT is Empirical . . . 19
2.2 Maximum Satisfiability . . . 21
2.3 Pseudo-Boolean Satisfiability . . . 22
2.4 Answer Set Programming . . . 22
2.5 Graphical Models and Generative Processes . . . 24
2.6 Conjugacy and the Dirichlet-Multinomial Pair . . . 26
2.7 Conclusions . . . 27

Chapter 3 Related Work **28**

3.1 Economics and Naïve Parallel Portfolios . . . 28
3.2 Algorithm Selection . . . 30
3.2.1 The Selection Problem . . . 30
3.2.2 Supervised Selection and `SATzilla` . . . 31
3.2.3 Other Supervised Portfolios . . . 33
3.2.4 Non-Model-Based Portfolios . . . 35
3.3 Solver Execution Scheduling . . . 37
3.4 Bandit Portfolios . . . 38
3.5 Domain-Focused Probabilistic Models . . . 39
3.5.1 Models of Solver Performance . . . 40
3.5.2 Models of Natural Language Text . . . 41
3.6 Conclusions . . . 42

Chapter 4 Modeling Solver Performance **43**

4.1 Run Time Distributions and Las Vegas Algorithms . . . 43
4.2 General Methodology . . . 45

4.3 A Family of Run Time Distribution Models 46
4.3.1 Maximum-Likelihood Multinomials 46
4.3.2 Multinomials with Uniform Smoothing 47
4.3.3 Multinomials with Non-Uniform Smoothing 48
4.3.4 Multinomials with Independent Mixture Smoothing 50
4.3.5 Multinomials with Linked Mixture Smoothing 51
4.4 Model Inference and Application 53
4.5 Comparing Models Empirically 54
4.6 Visualizing Model Output 56
4.7 Modeling for Exploratory Visualization 60
4.8 Conclusions 64

Chapter 5 Integrating Instance Features 66
5.1 Motivation 66
5.2 A Review of Instance Features 67
5.3 Feature Integration as Classification 70
5.4 Experiments 73
5.5 Conclusions 74

Chapter 6 Planning Solver Execution 78
6.1 Planning for a Known Run Time Distribution 79
6.2 Planning for an Uncertain Run Time Distribution 82
6.3 Experimental Evaluation 84
6.4 Conclusions 88

Chapter 7 Evaluation in SAT 89
7.1 Competition Benchmarks and Methodology 90
7.2 Evaluating Model Utility 92
7.3 Comparison with `ppfolio` 94

7.4 Comparison with SATzilla2009 97
7.5 Comparison with 3S 102
7.6 Conclusions 103

Chapter 8 Evaluations in PB, MAX-SAT, and ASP **110**
8.1 Comparisons in Pseudo-Boolean Satisfiability 111
8.2 Comparisons in Maximum Satisfiability 115
8.3 Comparisons in Answer Set Programming 120
8.4 Conclusions 125

Chapter 9 Discussion and Future Work **127**
9.1 The Modeling Component 127
9.2 The Feature Integration Component 130
9.3 The Scheduling Component 131
9.4 Parallelism and Beyond 133
9.5 Conclusions 135

Chapter 10 Conclusions **137**
10.1 Summary of Contributions 137
10.2 Closing Thoughts 139

Bibliography **141**

Vita **156**

List of Tables

2.1 An example SAT instance in CNF 12

3.1 Features used in `SATzilla` 32

5.1 Reduction in uncertainty for each category 73

7.1 Instance counts for SAT competitions 90
7.2 Instance features in the SAT domain 99

8.1 Instance features in the PB domain 111
8.2 Constituent solvers in a PB portfolio 115
8.3 Results of the 2011 PB solver competition 116
8.4 Constituent solvers in a MAX-SAT portfolio 118
8.5 Instance features in the MAX-SAT domain 118
8.6 Constituent `clasp` configurations in ASP 122

9.1 Example illustrating the suboptimality of planning 132

List of Figures

1.1 Outline of MAPP operation 7

2.1 Performance of SAT competition winners over time 18
2.2 Solvers' best performances in the 2009 SAT competition 20
2.3 An example graphical model: a mixture of Gaussians 24

3.1 Architecture of algorithm-selection portfolios 34

4.1 Run time distribution of `lingeling` 44
4.2 Log probability of test data under varied α 49
4.3 Plate diagrams of performance models 52
4.4 Log probability of test data, varying K 53
4.5 Log probability of test data, varying the model and sampling scheme 55
4.6 MAP estimates under each performance model 57
4.7 Inferred latent classes of solver performance 58
4.8 Inferred latent classes of solver performance 59
4.9 Component responsibilities over SAT 2007 instances 61
4.10 Categories of seriated SAT 2007 instances 62
4.11 Screenshot of the `borg-explorer` visualization tool 63

5.1 Run-success densities across variables/clauses 69

5.2 Reduction in uncertainty with feature set size 72
5.3 Feature coefficient weights across classifiers 75
5.4 Robustness of feature integration to feature quality 76

6.1 An example run schedule on SAT 2007 competition instances 79
6.2 Planner success rate at increasing bin counts 85
6.3 Plans computed over SAT 2011 competition categories 86
6.4 Planners' performance on SAT 2011 competition categories 87

7.1 Instances solved under various models 92
7.2 Portfolio performance relative to `ppfolio` 95
7.3 Plans employed versus `ppfolio` 96
7.4 Portfolio performance relative to `SATzilla2009` 98
7.5 Plans employed versus `SATzilla2009` 101
7.6 Portfolio performance relative to `3S` 104
7.7 Plans employed versus `3S` 106
7.8 Plans employed versus `3S`, continued 107
7.9 Plans employed versus `3S`, continued 108
7.10 Plans employed versus `3S`, continued 109

8.1 Performance on PB 2010 instances 114
8.2 Performance on MAX-SAT 2011 instances 121
8.3 Performance on `claspfolio` training instances 124

Acronyms

k-NN k-nearest neighbor. 34, 36, 74, 76, 93

AI artificial intelligence. 15, 17

ASP answer set programming. xi, 2–4, 9, 10, 15–17, 22, 23, 33, 34, 68, 105, 110, 118, 120, 125

BMC bounded model checking. 4, 17

BOW bag-of-words. 41

CDCL conflict-directed clause learning. 13, 14, 16, 18, 44, 79, 97, 103, 109

cdf cumulative distribution function. 71, 81

CNF conjunctive normal form. 12, 15, 16, 21, 23, 29, 31, 68, 113, 115, 118, 134

CP constraint programming. 36, 37

CPU central processing unit. 90, 94, 103, 116, 122

CV coefficient of variation. 99, 100

DPLL Davis-Putnam-Logemann-Loveland. 13, 14, 18, 23, 56, 68

EDA electronic design automation. 13, 16

EM expectation maximization. 6, 53, 56, 64

GMM Gaussian mixture model. 25

IDE integrated development environment. 17

IP integer programming. 37, 102, 116

KDE kernel density estimation. 69

KRR knowledge representation and reasoning. 17

MAP maximum a posteriori. xiii, 54, 57

MAPP modular architecture for probabilistic portfolios. 3, 6–11, 15, 22, 28, 36, 37, 43, 46, 60, 61, 64, 66–68, 70–74, 76–78, 82, 88–91, 93–110, 113, 114, 116, 117, 121–125, 127, 131, 133–138

MAX-SAT maximum satisfiability. xi, 3, 10, 16, 17, 21–23, 34, 105, 110, 115, 117, 118, 120, 121, 125, 130

MDP Markov decision process. 37, 38, 83

ML maximum likelihood. 24, 47, 53

MLN Markov logic network. 17

PB pseudo-Boolean satisfiability. xi, 3, 10, 16, 17, 22, 23, 105, 110, 111, 113–118, 120, 125, 130

PB-DEC pure pseudo-Boolean satisfiability. 22

PB-OPT linear pseudo-Boolean optimization. 22

PDE partial differential equation. 30

pdf probability density function. 24

QBF quantified Boolean formulae. 16, 35, 125

QCP quasigroup completion problem. 29, 40

RTD run time distribution. 6–9, 44–48, 50, 52–57, 63, 70, 71, 76, 78, 80, 82, 84, 88, 94, 111, 118, 129–131, 135, 138

SAT satisfiability. xiii, 2–4, 9–13, 15–23, 29–31, 34, 36, 37, 43–45, 54, 56, 57, 61, 67–69, 77, 84, 88, 91, 93, 105, 110, 111, 113–116, 118, 120, 125, 131, 133, 135

SLS stochastic local search. 13, 14, 18, 20, 36, 44, 56, 60, 68, 84, 85, 103, 105, 106

SMT satisfiability modulo theories. 15–17, 125

SVM support vector machine. 68

SVR support vector regression. 34, 120

UKP unbounded knapsack problem. 9, 80, 84, 133

XOR exclusive-or. 14

Symbols

A the number of knapsack items or actions. 80

B the number of modeled histogram bins. 46, 48, 71, 79–81, 84, 128

C processor time cutoff. 79

I the number of problem instances. 46–51, 83

K the number of latent classes in a model. 50, 51, 53, 55

M the number of similar distributions labeled in classification. 71, 131

R the number of runs made by some solver on an instance. 46–51

S the number of solvers in a suite. 46–51, 71, 80, 81, 83

T the number of actions in a solver execution schedule. 79, 83

V the number of variables in a problem instance. 12

Δ the set of points on a simplex. 26

$\boldsymbol{\alpha}$ an arbitrary vector of reals. 26, 27, 49–51

$\boldsymbol{\beta}$ an l_1-normalized vector of reals. 24, 25, 50, 51, 54, 60, 71

$\boldsymbol{\phi}$ a vector of reals representing instance features. 70

$\boldsymbol{\theta}$ an l_1-normalized vector of reals. 7, 26, 47–51, 71

δ the Kronecker delta. 47

$\mathbb{Z}$ the set of integers. 26

$\mathcal{N}$ the normal (Gaussian) distribution. 25

Dir the multinomial distribution. 26, 27, 48–51

Mul the multinomial distribution. 25, 26, 47–51

Chapter 1

Introduction

When humans solve a problem for the first time, they learn from the experience of solving it. They uncover patterns in the problem, they create new techniques to apply, and they weigh the cognitive effort required by each technique. Humans also learn that a particular solution strategy worked once, and that it may work again.

Such knowledge is valuable because a problem is rarely the only one of its kind. As an airline chooses its flights, a school schedules its courses, or a hospital assigns employees to shifts, many instances of the same type of problem must eventually be solved. As humans solve these instances in sequence, the experience they gain allows them to improve their solutions and increase their productivity over time. Even when humans solve problems by leveraging computational tools, their presence in the problem-solving loop adds value because of this ability to learn from experience. They learn to mitigate software quirks, to make judgments about problem difficulty, to manipulate problem representations to better suit their tools, and, crucially, to select the tools themselves, identifying those that perform best on the problem at hand.

Can human guidance be removed from the problem-solving loop? Imagine a system that adapts to the problems it sees, learning to solve them better over time, as humans do. One important aspect of such adaptation is learning to use the tools that are best on

the problem at hand. The area of *algorithm portfolios* develops systems that attempt to learn in this way. Portfolios automatically choose which computational tools to apply and when to apply them, tailoring those decisions to each problem. This dissertation introduces a new type of algorithm portfolio that, among other advantages, can better decide when a constituent tool will benefit from multiple attempts to solve the same problem. This chapter explains the motivation behind its development and behind the development of algorithm portfolios in general, then outlines the ideas presented in this dissertation and the new capabilities that they provide.

1.1 Motivation

Heuristic algorithms for logical reasoning have become indispensable tools in science and industry. These *solvers* are increasingly successful on large instances of computationally difficult problems such as satisfiability (SAT) and answer set programming (ASP), and lie at the heart of applications from circuit verification (Clarke et al., 2001) to software synthesis (Erkök et al., 2009), among many others (Naveh, 2010). The research progress in this area has been remarkable and sustained. In SAT, for example, instances of more than several hundred variables were considered out of reach less than twenty years ago (Selman et al., 1993). Today, solvers routinely analyze hardware designs encoded as propositional formulae involving millions of variables (Kautz and Selman, 2007).

Whether a problem instance can be solved, however, often depends in practice on whether the correct solver was selected and its parameters appropriately set. The profusion of successful new solvers has not been accompanied by successful new formal techniques for understanding and predicting their behavior. Worse, each solver has strengths and weaknesses; none dominates empirical comparisons. The task of choosing a method to apply to a given problem instance, what Rice (1976) termed the *algorithm selection* problem, is thus both important and difficult. In practice, selection decisions are frequently subjective and suboptimal.

Algorithm portfolios (Huberman et al., 1997) are a general class of methods for automating such decisions. Based on solvers' past performance and an instance's superficial appearance, they allocate computational resources and attempt to maximize the fraction of those resources spent on an algorithm well-suited to that instance. Portfolios thus reduce human effort. By analyzing problem instances more thoroughly, solver performance more comprehensively, and allocation options more broadly than can a human user, they are also able to improve overall performance in ways that humans cannot.

This dissertation presents a novel and effective portfolio approach: the modular architecture for probabilistic portfolios (MAPP). It is built around the estimation, prediction, and exploitation of complete solver performance distributions. It includes new techniques for core aspects of portfolio construction and operation, and exhibits state-of-the-art performance as a whole, including first-place finishes in two consecutive years of the pseudo-Boolean satisfiability (PB) solver competition (Manquinho and Roussel, 2010, 2011). Its modular design is motivated by the decomposition of an algorithm portfolio into three fundamental components. First, run time distributions are estimated from limited data. Second, similar distributions are identified according to instance appearance. Third, solver execution schedules are computed for weighted mixtures of such distributions. The development of each of these components, along with the MAPP approach as a whole, are the key contributions of this dissertation.

1.2 Why Algorithm Portfolios?

This dissertation focuses on four problem domains in its evaluations: SAT, ASP, PB, and maximum satisfiability (MAX-SAT). These domains include valuable computational questions, such as finding errors in a circuit or bugs in a program. These problems and their importance are discussed further in Chapter 2.

From a worst-case perspective, solving these problems is intractable; solving them in practice thus relies on heuristics. While these heuristics often succeed, intuition and

experience must guide how and when they are applied. Writing about a widely-used ASP solver, Gebser et al. (2011b) noted that they "are unaware of any true application on which `clasp` is run in its default settings". Continuing along these lines, Gebser et al. (2011a) wrote that "choosing the right parameters often makes the difference between being able to solve a problem [and] not".

These statements can be applied to many other domains. For example, formal verification plays a vital role in processor development at Intel Corporation, where bounded model checking (BMC) based on SAT has been particularly successful at "targeted bug hunting" (Kaivola et al., 2009). The custom SAT solver developed by Intel, named `Eureka`, has been invoked millions of times by hundreds of different projects. To maintain robustness against this heterogeneous set of inputs, however, they implement a variety of interchangeable heuristics, and draw on years of human experience to apply specific configurations to particular domains (Nadel, 2011). Modern algorithm portfolio systems seek to acquire, from data, some of the benefits of such experience. They may, therefore, benefit any user needing to apply a tool for logical reasoning to many different inputs.

One motivation for the development of portfolio methods is therefore pragmatic: they address a demonstrated practical need, one shared by users of heuristic solvers across many different domains. Another motivation is idealistic. Although empirical evidence, rather than analytical knowledge, is typically required to apply heuristics effectively in problem solving, that evidence should nonetheless be applied in a principled way. An algorithm portfolio system codifies a set of principles for leveraging empirical evidence in solving computational problems with the aid of heuristics.

1.3 General Portfolio Operation

Any portfolio method is built around a *solver suite*: a small, finite set of arbitrary algorithms that take any *instance* of some problem domain as input. These solvers may be different algorithms, different parameterizations of the same algorithm, or some mixture of the two.

After executing for some *run time*, potentially infinite or nondeterministic, a solver may return a solution to the instance; if a solution is returned, it is guaranteed to be correct. In practice, computational resources are constrained, and the length of a solver's run is limited to some *cutoff* value.

A portfolio acts as a solver like any other. It takes an instance as input, is subject to a run time cutoff, and may or may not return a solution under those constraints. Unlike a typical solver, however, it does not attempt to solve the instance directly. Instead, it acts through its suite of solvers, passing them the instance to be solved and allocating processor time to different solvers according to the strategy it employs. Members of a portfolio's solver suite are often referred to as *subsolvers*.

Portfolio methods succeed when they are able to exploit patterns in solver performance specific to a particular distribution of problem instances. To find such patterns, they analyze performance on training instances assumed to be drawn from the same distribution they will encounter in testing. Data collection on those training instances may entail running every solver on every instance. Portfolio training is, therefore, usually an offline process.

Simple portfolio strategies are often employed manually by practitioners, without explicitly naming them as such. One example is the common practice of evaluating multiple solvers on a set of instances in order to select the single solver with the best observed performance, to be used thereafter. This strategy will be used as a baseline in many of the experiments to follow.

More sophisticated portfolio strategies take advantage of *instance features*, i.e., efficiently-computable properties of an instance that may offer clues to solver performance. The `SATzilla` and `claspfolio` systems, for example, use the features of an instance to predict the expected computational cost for each solver of obtaining a solution, then execute the one solver believed likely to incur the least cost (Xu et al., 2008a; Gebser et al., 2011b). However, like most existing methods, they treat solvers as effectively deterministic,

and restrict themselves to the algorithm selection problem. The following section presents a high-level picture of MAPP, the more general approach developed in this dissertation.

1.4 The Modular Architecture for Probabilistic Portfolios

The fundamental principle of MAPP is to represent portfolio belief in terms of a subjective discrete probability distribution over solver performance distributions. An individual solver exhibits a fixed distribution over possible run times on a given problem instance; a suite of solvers exhibits multiple distributions on an instance, one for each solver it contains. A solver suite and a distribution over possible problem instances thus define, together, a distribution over such groups of run time distributions (RTDs). MAPP represents its knowledge about this distribution over distributions, acquired in training, as a finite set of possible groups of RTDs, weighted by their probabilities.

MAPP is a modular architecture. Each of its three high-level components is responsible for an essential task in manipulating or leveraging its subjective belief representation. First, in training, a *solver performance model* establishes the portfolio's prior belief. The parameters of this model are fit to observed training runs with expectation maximization (EM), and form estimates of the RTD of each solver on each training instance. The uniform distribution over per-instance groups of RTDs becomes the initial belief. Second, at test time, information from domain-specific features is leveraged using a collection of supervised classifiers. These *distribution similarity models* predict training instances on which solvers' performance is likely similar to their performance on the test instance. These predictions are used to reweight the portfolio's belief, and the result becomes its conditional belief about solver performance on a specific instance. Third, a *solver execution planner* computes a sequence of solver runs tailored to the portfolio's conditional belief. The goal of this sequence is to maximize the probability that one of its runs will succeed.

This architecture is designed to keep the concerns of its components separate while enabling good performance. Portfolio belief, for example, is represented as a weighted,

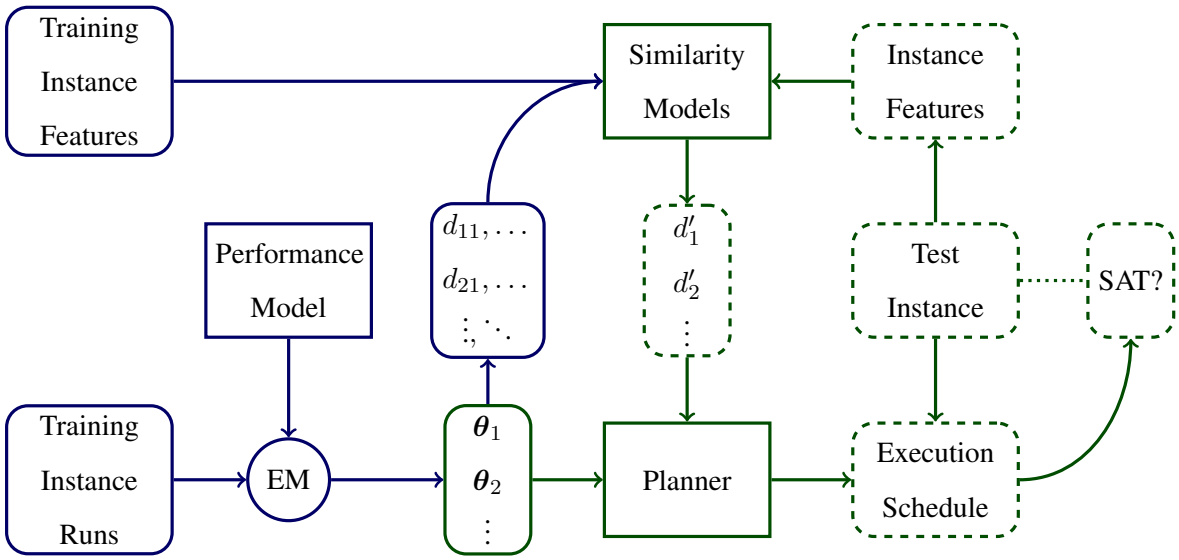


Figure 1.1: The operation of MAPP, the modular architecture for probabilistic portfolios. The elements of the architecture that are active only during the training phase are colored blue; the elements that are active during the test phase are colored green. Dashed borders indicate values that change for each test instance. The overall structure is centered around a representation of uncertain belief as a set $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots\}$ of possible RTDs. This set is established in training by a performance model. It is reweighted for each test instance by a collection of supervised models that each predict its inter-RTD similarity d'_i based on appearance. An execution schedule is then computed and implemented based on this conditional belief. Unlike existing portfolio methods, MAPP is thus able to make multiple runs of multiple solvers on a single instance, when doing so makes success more likely.

finite set of possibilities because planning can be performed efficiently under that belief representation. Feature integration is treated as classification so that this belief representation can be maintained while building on the mature techniques of supervised learning. The explicit solver-performance modeling step is motivated by the expense of acquiring training data; it can extract information even when not every solver is run on every training instance.

Before this dissertation expands on each component in the chapters to come, it is helpful to review MAPP as a sequence of concrete steps carried out in two distinct phases. During the training phase,

1. a suite of solvers is assembled;
2. each of those solvers is run zero or more times on each training instance;
3. observed run times are used, in combination with a probabilistic model of solver performance, to estimate the complete distribution over run times for each solver on each instance;
4. a matrix of pairwise distances between these RTDs is computed using an appropriate measure of distance;
5. a vector of superficial features is computed for each training instance; and
6. discriminative models, one for each RTD, are trained to predict the similarity between pairs of RTDs based on the features of their associated problem instances.

After training, the portfolio is ready to solve new instances. During the test phase that follows,

1. a previously-unobserved problem instance is received, and is assumed to be drawn from the same distribution as were instances during training;
2. its superficial features are computed;
3. the discriminative models constructed during training are used to predict the similarity between solvers' unknown RTDs on the new instance and their estimated RTDs on each training instance;
4. the mixture over training-instance RTDs is reweighted to favor more similar instances, and this mixture represents the portfolio's subjective belief about solvers' possible RTD on the new instance;

5. a dynamic-programming *planner*, based on a reduction to the unbounded knapsack problem (UKP), computes a schedule of solver runs to maximize the probability of success based on that subjective belief; and

6. the portfolio carries out that plan.

The entire process is illustrated in Fig. 1.1. This dissertation proceeds according to the organizational lines suggested by the architecture itself, as the next section outlines.

1.5 Outline of the Dissertation

Chapter 2 describes the computationally difficult problem domains targeted in this work, including SAT and ASP. It discusses the history and direction of solver development in these areas, and motivates their position as the focus of algorithm portfolio research. It also reviews the foundations of probabilistic graphical models and the discrete distributions employed by this dissertation.

Chapter 3 surveys the wide array of existing research on portfolio methods, including prominent, successful systems such as `SATzilla` (Xu et al., 2008a). It describes the strengths and weaknesses of these architectures, and places MAPP in the context of the field as a whole. Connections to other relevant work, such as the emergence of complex hierarchical models of natural language text, are also described.

Chapter 4 focuses on the first module of MAPP. It develops a family of unsupervised mixture models of solver performance. It uses these models to estimate, based on a varying number of runs, complete distributions over run time for standard solver suites on competition benchmarks. The models are then compared according to the accuracy of their predictions.

Chapter 5 constructs the second module of MAPP. It applies supervised learning to leverage instance features in this setting, using a standard classifier but maintaining the mixture-of-RTDs belief representation central to this architecture. It quantifies the value of

such features in terms of modeling accuracy, and analyzes the predictive role of individual features belonging to a standard set.

Chapter 6 introduces the third module of MAPP. It examines the problem of planning solver execution under a weighted set of possible solver performance scenarios. It expands on the transformation of solver scheduling into a knapsack problem, and uses this connection to apply dynamic programming. The resulting algorithm schedules solvers optimally in the case of certainty and approximately in the case of uncertainty.

Chapter 7 then combines the techniques of the preceding three chapters into the full MAPP framework. Replicating the SAT solver competition environment to simulate head-to-head-comparisons, it contrasts MAPP empirically with three of the most successful existing portfolio methods. Chapter 8 extends these comparisons to PB, MAX-SAT, and ASP.

Chapter 9 and Chapter 10 consider the dissertation in hindsight. The former suggests directions for future work, and the latter summarizes the primary contributions of the MAPP framework: a new direction in algorithm portfolio research, and an effective problem-solving tool.

Chapter 2

Background

This chapter covers background material relevant to the development of the MAPP and the techniques it employs. This review spans both the areas such as SAT to which it is applied, and the conceptual tools such as probabilistic graphical models and maximum-likelihood estimation from which it is fashioned. Furthermore, the chapter motivates some of the choices made: Section 2.1, in particular, justifies the selection of SAT and its family of related problems as the focus of portfolio development.

2.1 Satisfiability

This dissertation focuses primarily on the computationally difficult problem of SAT, which asks whether any assignment of truth values to the variables of a given Boolean formula can make that formula true. SAT is the classic NP-complete problem (Cook, 1971), often serves as a target encoding for problems of interest in application-specific areas (Gomes et al., 2008), and has long played an important role in algorithm portfolio research (Gomes and Selman, 1997a). This section first reviews SAT and its solution methods, then makes and supports three specific claims: first, that solving SAT is useful, both to industry and to other research communities; second, that solving SAT is difficult, but not so difficult as to

	Clause						
	(	x	$\vee$	y			)
$\wedge$	(			$\neg y$	$\vee$	$\neg z$	)
$\wedge$	(	$\neg x$			$\vee$	$\neg z$	)
$\wedge$	(	$\neg x$	$\vee$	y	$\vee$	z	)

Table 2.1: A small example instance of SAT, i.e., a Boolean formula in CNF. This instance happens to be satisfiable: it is satisfied by $x \wedge y \wedge \neg z$, among other assignments. Inclusion of the unit clause (z), however, would render the formula unsatisfiable.

be uninteresting; and, third, that effectively solving SAT is often an empirical question—and that both developers and practitioners thus need a principled decision-making layer to handle some of those empirical concerns. Taken together, these claims motivate SAT as the primary testbed for portfolio research, both in this dissertation and in much of the related literature.

2.1.1 SAT and Its Solvers

This review begins by covering the basic structure of SAT and SAT algorithms. The Boolean formula representing an instance of the SAT problem is almost always expressed as an "and" ($\wedge$) gate, or conjunction, over groups of "or" ($\vee$) gates, or disjunctions. This scheme is termed conjunctive normal form (CNF). The instance presented by Table 2.1, for example, is a conjunction of four *clauses*. Each clause consists of some number of *literals*—symbols representing Boolean variables, optionally negated—joined by $\vee$s.

Does any assignment of truth values to the V variables in this formula make it evaluate to true? Naïvely, establishing the answer to this question involves generating and testing every possible such assignment, of which there are 2^V. From a worst-case perspective, of course, no subexponential-time approach is known. In practice, however, formulae of inter-

est often have common structural properties that can be exploited by a solver implementing an appropriate heuristic.

The methods employed by modern SAT solvers are the fruit of a long and active research history. Seminal work includes that of Selman et al. (1994) on stochastic local search (SLS), of Marques-Silva and Sakallah (1999) on conflict-directed clause learning (CDCL), and of Gomes et al. (1998) on solver randomization. Many of these solvers derive from the venerable Davis-Putnam-Logemann-Loveland (DPLL) procedure introduced by Davis et al. (1962). Three aspects of SAT are responsible for much of how its solvers behave, and are therefore important to the design and effectiveness of SAT portfolios. First, most problem instances can be placed into one of three broad categories:

- *random* instances sampled from, e.g., formulae near the phase transition in the clause-variables ratio (Mitchell et al., 1996) or from encoded random partial latin squares (Gomes and Selman, 1997b);
- *crafted* instances encoding hand-selected synthetic problems such as parity checking or puzzles such as the Towers of Hanoi; and
- *industrial* instances contributed by organizations using SAT solvers to tackle problems of commercial value, such as verification and electronic design automation (EDA).

Second, the set of SAT solvers can be divided roughly into two families:

- *complete* solvers, which almost always implement DPLL-derived systematic tree search, and can prove both satisfiability and unsatisfiability; and
- *incomplete* local-search solvers, which take a biased random walk through the space of variable assignments, and can prove only satisfiability.

Understanding the general operation of these solver families is important to understanding the sources of the performance diversity that allows algorithm portfolios to succeed.

Complete DPLL-derived solvers operate by repeatedly selecting a variable on which to branch, guessing a value for that variable, and backtracking if a conflict is detected (Gomes et al., 2008). They build, crucially, on this straightforward recursive foundation by greedily applying, at every decision step, the operations of *unit propagation*, in which variables are assigned so as to satisfy all single-literal clauses, and *pure literal elimination*, in which variables whose literals share a common polarity are set according to that polarity. The critical decisions in such a solver, then, are the choices of variable on which to branch and of truth value for that variable. CDCL solvers add additional logic to the DPLL foundation by automatically augmenting the formula with "conflict" clauses representing new learned constraints among variables. The benefit provided by such clauses, in the early detection of barren search avenues, must be weighed against their cost, in the execution overhead incurred on larger formulae. Clause-learning solvers, then, make other important decisions regarding how and when to learn such clauses, as well as how and when to select learned clauses to discard.

Incomplete SLS solvers operate by repeatedly updating a global assignment of truth values to variables, according to some randomized procedure. The random walk thus taken through the space of assignments is biased toward minimizing the number of clauses left unsatisfied. In the classic `WalkSAT` procedure, for example, the update procedure focuses on variables appearing in unsatisfied clauses (Selman et al., 1994). Most SLS solvers modify only one variable at a time, and the central decision in such solvers is the selection of that variable. This choice, however, is often decomposed: the solver may first choose a clause on which to focus, or it may incorporate some random-step probability balanced against the overall search bias.

In general, complete solvers outperform incomplete solvers dramatically on highly-structured industrial instances. These solvers may also recognize and exploit common patterns in particular application domains. The solver `CryptoMiniSat`, for example, implements special handling of the exclusive-or (XOR) constraints common to cryptographic

problems (Soos et al., 2009). At the same time, incomplete solvers vastly outperform systematic methods on less-structured randomly-generated instances, and can often perform well on satisfiable instances of hand-crafted challenge domains (Järvisalo et al., 2011a). Among the members of a single solver family, analogous divisions exist: the complete solvers `march_hi` and `lingeling`, for example, were developed for different reasons and exhibit wildly different performance characteristics.

Such performance diversity drives the need and creates the opportunity for portfolio methods like that presented by this dissertation. The next several sections dig more deeply into the causes of this diversity. They cover the motivation behind solving SAT in practice and behind the development of new SAT solvers, the success and struggle of solver algorithm research, and the inherent empirical component to algorithm development.

2.1.2 Solving SAT is Important

Why does anyone care about solving SAT? Solvers, after all, cannot promise polynomial-time solutions to worst-case instances. Instead, they offer only solutions to *some* instances, *some* of the time. In light of that limitation, who and what drives the manifest research interest in better solvers? Are methods that amplify solver performance, such as MAPP, valuable? This section presents three motivations for improving SAT solver performance: first, the value of improved performance in SAT to solvers for strongly-related problem representations, such as satisfiability modulo theories (SMT) and answer set programming (ASP), which may employ SAT solvers or methods at their core; second, the value to numerous applications in science and industry; and, third, the value to other areas of research in artificial intelligence (AI).

SAT is among the simplest constraint satisfaction problems. Its core problem representation, propositional formulae in CNF, is primitive: instances consist of identical variables, locked within the rigid cage of normal form. SAT, however, sits at the center of a constellation of computationally difficult problems that impose fewer restrictions. Two is-

sues have spurred interest in these more general forms. First, some important application domains are not easily expressed as decision problems, such as those involving optimization or maximization. Second, higher-level representations can be easier for humans to express and to understand.

Examples of non-decision relatives of SAT include the NP-hard MAX-SAT problem and its weighted and partial variants. Examples of SAT relatives permitting more expressive decision-problem representations include the problem of SMT, the problem of quantified Boolean formulae (QBF), and the problem of pseudo-Boolean satisfiability (PB). Specific applications often drive interest in these areas. Arithmetic operations in a hardware verification problem, for example, can be expressed in terms of bit vectors by SMT (Bruttomesso et al., 2007). Common higher-level constraints such as cardinality restrictions can be expressed more concisely in PB than they can in propositional CNF (Dixon and Ginsberg, 2000).

Of these difficult computational problems, this dissertation focuses on three in addition to pure SAT: MAX-SAT, PB, and ASP. Sections 2.2 to 2.4 lay out the relevant properties of these domains. Despite their apparent differences, however, these areas benefit directly from improvements to SAT, either because their instances are solved after transformation into CNF, or because new solver techniques that prove promising in SAT are adapted by their solvers. Examples of the former include the `Cmodels` solver for ASP (Lierler and Maratea, 2004; Giunchiglia et al., 2006) and the `MiniSat+` solver for PB (Eén and Sörensson, 2006). Examples of the latter include the `clasp` solver for ASP, which successfully applies the techniques of CDCL to that domain (Gebser et al., 2007).

Success in solving even some instances of these difficult computational problems is important because these instances often represent individual questions with business or scientific value. In some cases, these problems are expressed directly in CNF. Planning was one of the first (Kautz and Selman, 1992), and was followed by problems in verification (Biere et al., 1999; Clarke et al., 2001), EDA (Wood and Rutenbar, 1997; Veneris, 2003),

configuration management (Batory, 2005), and many others (Gomes et al., 2008). Some application areas therefore benefit immediately from improved SAT methods.

The constellation of SAT relatives covers an even wider range of applications, many of which see regular commercial use. BMC using SAT was successfully employed during the development of the Intel Core i7 (Kaivola et al., 2009). The `Z3` solver for SMT is used to verify device drivers for Microsoft Windows 7 (de Moura and Bjørner, 2010, 2011). Component dependencies in the Eclipse integrated development environment (IDE), the most widely-used Java IDE in the world, are managed as PB constraints with the solver `sat4j` (Le Berre and Rapicault, 2009). Matches in the Eredivisie and Eerste Divisie, the top soccer leagues of the Netherlands, are scheduled by the SMT solver `Barcelogic` (Nieuwenhui, 2009). It is clear that the methods in the SAT constellation include some of the most significant success stories for automated logical reasoning.

The significant commercial value of improved SAT tools is coupled with significant value for research in other areas of AI. Research in knowledge representation and reasoning (KRR), for example, often relies on the existence of effective methods for ASP (Gelfond and Leone, 2002; Baral, 2003), which in turn build on approaches from SAT. Inference tools for probabilistic reasoning have employed SAT and MAX-SAT techniques at their core (Park, 2002), especially those for frameworks, such as Markov logic networks (MLNs), that integrate the languages of logic and probability (Gogate and Domingos, 2011).

2.1.3 Solving SAT is Difficult (But Not *Too* Difficult)

The success of modern methods, however, raises the danger of overstating their power: SAT remains, and is likely to remain always, a problem for which our demand for solutions outstrips our ability to provide them. Even seemingly trivial problems, such as proving the unsatisfiability of small instances of the so-called "pigeonhole principle", are topics of ongoing research (Audemard et al., 2010). Dozens of solvers, representing the state of the art, running on modern hardware, each dedicated 5,000 s of processor time to every one of

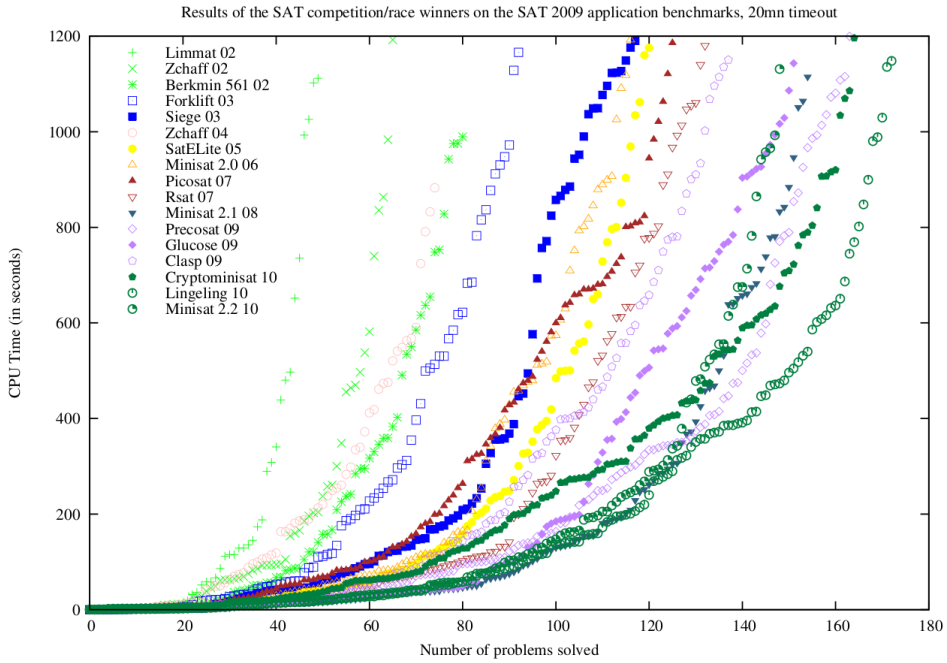


Figure 2.1: The performance of every SAT competition winner in the INDUSTRIAL category running on a common benchmark collection and modern hardware; these data were provided by Le Berre (2011). The year in which each solver won is noted. Algorithmic improvements alone have yielded dramatic and consistent improvements in performance over time, rendering instances tractable that were previously far out of reach. Impressive, but also indicative of the empirical underpinnings of much of this progress, is that the progress depicted has come *after* the introduction of efficient CDCL, considered the key modern addition to DPLL (Gomes et al., 2008).

1,200 instances in the 2011 SAT competition; in the end, however, more than 200 of those instances remained unsolved by *any* solver (Järvisalo et al., 2011b).

Research in SAT methods therefore sits between future challenge and past success. Since the surprising effectiveness of early SLS solvers (Selman et al., 1992), algorithmic gains in efficiency on instances of interest have been consistent and substantial. Figure 2.1

plots the performance on modern benchmarks of every winner in the INDUSTRIAL category since the inaugural SAT competition ten years ago. Over the past decade, ignoring gains from hardware, solvers have gained the ability to solve entire classes of instances that were previously intractable. This remarkable achievement fuels a virtuous cycle of success driving interest driving success.

This trajectory will continue, however, only if the research community continues to discover new ways to exploit the structure of important instances. This pressure may accelerate the process of specialization typified by solvers like `CryptoMiniSat`. In this direction, Karp (2011) has proposed an explicit methodology for heuristic algorithm development that focuses on specific application areas, with a strong focus on empirical evaluation. Data-driven specialization in algorithm development may further increase demand for data-driven decisions in algorithm selection.

2.1.4 Solving SAT is Empirical

Solvers are fundamentally heuristic, and their heuristic nature means that understanding their behavior is, in practice, an empirical enterprise. The lead author of `MiniSat` (Eén and Sörensson, 2004), a modular solver used as the basis of much solver research, has stated that "SAT is an experimental science" (Eén, 2011). Solver development is shaped by a reliance on empirical results. As different solver authors create and evaluate heuristics in a closed loop on different benchmark instances, corresponding differences in the empirical feedback they receive cause differences to emerge in their solvers' performance characteristics.

Solvers are therefore often complementary. While obvious qualitative distinctions exist between solver families—complete solvers tend to outperform local search solvers on more structured industrial instances, while they tend to underperform them on less structured random instances—no solver is unequivocally best even within a family. Figure 2.2 summarizes the number of problem instances on which each solver performed best in the 2009 comprehensive solver evaluation. Note that every solver performs best on some non-

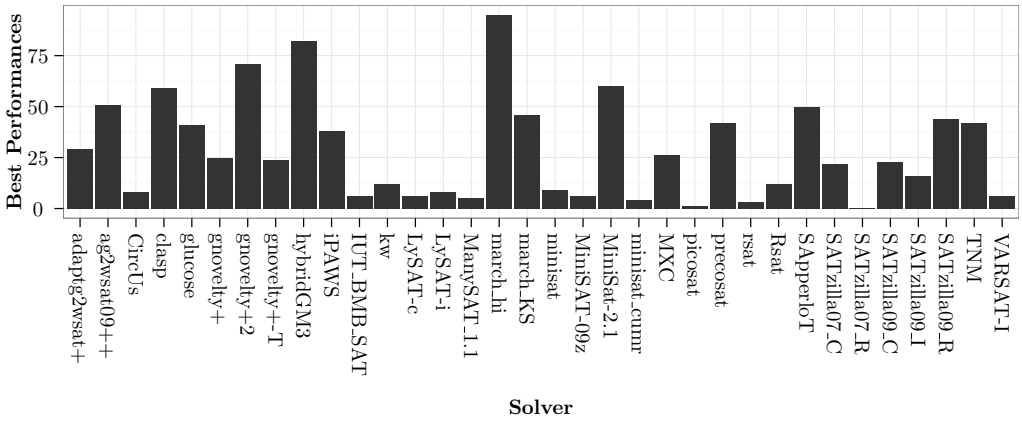


Figure 2.2: The number of instances on which each solver performed best in the 2009 SAT competition (Le Berre et al., 2009). Note that no solver is dominated completely. The apparent exception at the far right is a 2007 version of the `SATzilla` solver, discussed in Section 3.2.2, which is limited to the best performance of its older constituent solvers. It was included in the competition as a point of reference.

empty subset of the benchmark collection.

The connection between heuristics and specialization has been exploited by recent "highly parameterized" methods, which implement multiple heuristic variants and intentionally expose numerous parameters. These parameters are then tuned, by an automated method, for a particular distribution of instances. The SLS solver `Captain Jack`, for example, implements several dozen separate parameters that can dramatically—and unpredictably—affect its performance on different instance classes (Tompkins et al., 2011). Nikolić et al. (2009) and Hutter et al. (2007) obtained similar results with analogous solvers in the DPLL family.

SAT is thus an appealing testbed for portfolio research. Solver behavior is complex, and requires empirical analysis to evaluate. Improvements to SAT methods have practical

and scientific value. The prominence of solver competitions leads to the open distribution of solver packages, solver source, and comprehensive collections of challenging problem instances in a standard format. For these reasons, many portfolio methods have been connected to SAT, such as the naïve parallel portfolios described in Section 3.1 and the particularly successful SATzilla portfolio described in Section 3.2.2. It is an obvious target domain for continued work on portfolio methods, and will be the central focus of this dissertation. Generality, however, is an important goal of this research as well; the next several sections describe the relevant details of other domains that will be evaluated.

2.2 Maximum Satisfiability

An instance of MAX-SAT, like an instance of SAT, is a CNF expression. Rather than searching for a globally-satisfactory assignment, however, a MAX-SAT solution is the *maximum* number of clauses that can be satisfied by any variable assignment. Expressed as a decision problem, MAX-SAT considers whether a formula exists that satisfies *at least* some number of clauses in the formula. By assigning a weight to each clause, and requesting the assignment maximizing the total weight of satisfied clauses, several problem variants can be expressed:

- unweighted MAX-SAT, in which every clause receives equal weight;
- weighted MAX-SAT, in which some clauses receive different weight;
- partial MAX-SAT, in which certain clauses may not be left unsatisfied, while the remaining "soft" clauses receive equal weight; and
- weighted partial MAX-SAT, identical to partial MAX-SAT except that the soft clauses may receive different weights.

No algorithm portfolio has participated in recent MAX-SAT competitions, so the comparisons in this paper will be made against individual solvers and alternative baseline portfo-

lios.

2.3 Pseudo-Boolean Satisfiability

The problem of PB considers a more general class of formulae: conjunctions of inequality constraints on first-degree polynomials, in which variables are constrained to be "pseudo-Boolean", i.e., to belong to the set $\{0, 1\}$. This problem is also known as "0-1 programming" or "binary integer programming" in the linear programming community (Eén and Sörensson, 2006). As in MAX-SAT, several important problem variants exist, each of which will be addressed in this dissertation:

- pure pseudo-Boolean satisfiability (PB-DEC), a decision problem asking only whether any assignment of values to variables can satisfy a formula's constraints;
- linear pseudo-Boolean optimization (PB-OPT), an optimization problem which seeks the satisfying assignment of values to variables that also minimizes the value of a given objective function; and
- nonlinear versions PB-DEC-NLC and PB-OPT-NLC of these problems, in which the constraints may also include product terms.

In the most recent PB solver competitions, held in 2010 and 2011, precursors to MAPP took first place in the main SMALLINT-DEC-LIN category under the system name `borg`. This dissertation generalizes the approach used to win these competitions.

2.4 Answer Set Programming

MAX-SAT and PB can be viewed as results of efforts to make SAT more expressive. The field of ASP, however, may be best viewed as the result of making an expressive problem representation—declarative logic programming in the style of Prolog—more specialized, with the goal of enabling efficient reasoning and its application to difficult combinatorial

search problems (Lifschitz, 2008). Unlike CNF expressions in SAT, which are almost always too large to be written by hand, a problem can be specified in ASP concisely. As an example, Lifschitz (2008) expressed the classic problem of clique-finding, in this case looking for cliques of at least ten vertices, in only two statements:

```
10 {in(X) : vertex(X)}.
:- in(X), in(Y),
   vertex(X), vertex(Y), X != Y,
   not edge(X, Y), not edge(Y, X).
```

Representing such a problem directly in CNF involves understanding and considering possible encoding strategies, then implementing and testing an imperative program to generate formulae appropriately. The formula for any graph of more than a few vertices would not fit on this page. The expressivity of ASP can therefore be appealing in applications with a substantial modeling component, such as the management of spacecraft operations (Nogueira et al., 2001) and the reconstruction of phylogenetic trees in linguistics and zoology (Brooks et al., 2007).

Before solving an answer set program, it is "grounded", i.e., it is transformed so that its variables are eliminated. As discussed in Section 2.1.2, some instances can be translated entirely into CNF and solved as SAT. Dedicated ASP solvers also apply many of the methods pioneered in SAT, such as clause learning and DPLL search.

This dissertation aims to make SAT, MAX-SAT, PB, and ASP solving more efficient, and, to do so, it is important to understand the landscape of these methods and the problems they are attempting to address. The primary techniques employed by this work, however, are not those of SAT, but those of Bayesian statistics. The following sections briefly review these fundamental techniques.

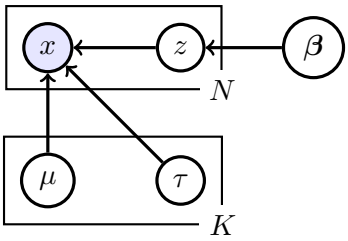


Figure 2.3: Graph representation of an example model, a standard GMM. Nodes represent the model's variables; shaded nodes are observed, and unshaded nodes are latent. Arrows represent conditional dependencies. Boxes mark sections of the model that appear multiple times, and the value associated with each box specifies the number of times the contents of that box are duplicated. In this case, each of N data are drawn from one of K normal distributions, where the index of z of each datum is drawn from a multinomial distribution parameterized by $\boldsymbol{\beta}$.

2.5 Graphical Models and Generative Processes

Graphical models have become a core framework for Bayesian probabilistic reasoning (Koller and Friedman, 2009). The primary task in such reasoning is to make principled judgments about what is unobserved, or *latent*, based on what is observed. Maximum likelihood (ML) estimation is the standard objective in this task: given a set of observations, and a set of statistical assumptions about the process giving rise to those observations, i.e., a model, what is the most likely unobserved state of that process? If those assumptions are sufficiently well-behaved, such as that the data are drawn from a single normal distribution, ML estimation is straightforward. In some cases, for example, latent variables can be determined analytically after setting the first derivative of the log probability density function (pdf) to zero. As assumptions grow more complex, inference becomes intractable—unless the larger model is carefully structured.

In particular, inference remains possible when the dependencies between variables

are minimized, allowing the joint distribution of the entire model to be *factorized* into groups of related variables, each of individually tractable form. *Graphical models* are an approach to representing these dependencies. In the directed graphical models used here, one example of which is shown in Fig. 2.3, edges denote conditional dependencies. Such a graph is therefore a partial representation of a data-generating, or *generative*, process, a representation that focuses on the important relationships between variables. In the notation used by this dissertation, the simple Gaussian mixture model (GMM) of Fig. 2.3 may be specified fully by:

$$z_n \sim \text{Mul}(\boldsymbol{\beta}), \qquad n \in 1 \dots N, \tag{2.1}$$

$$x_n \sim \mathcal{N}(\mu_{z_n}, \tau_{z_n}), \qquad n \in 1 \dots N. \tag{2.2}$$

Less concisely, to assume that the data were drawn from this generative process is to assume that

- the N observed data were each drawn from one of K normal distributions, each parameterized by an unknown mean μ_k and precision τ_k, and that
- the index conveying the component associated with each datum was drawn from a shared multinomial distribution, parameterized by $\boldsymbol{\beta}$.

This combination of graph representation and process specification states the structure of a model and its underlying statistical assumptions clearly.

Graphical models are thus an approach to understanding and building complex but tractable joint distributions. Their flexibility allows wide latitude in choosing the distributions of internal latent variables. This dissertation makes good use of the two workhorse distributions of discrete models, the Dirichlet-multinomial pair, described next.

2.6 Conjugacy and the Dirichlet-Multinomial Pair

Take a set of latent parameters of some distribution, and assume that they are drawn from some other *prior* distribution. The parameters of the prior specify our initial beliefs about—i.e., our subjective probabilities regarding—the values of the latent parameters drawn. If some number of draws from the unknown distribution are observed, our beliefs about its latent parameters should typically change to reflect that new information. The distribution representing these data-conditioned beliefs is called the *posterior*. For certain so-called *conjugate pairs* of distributions, the prior and the posterior share the same form. The graphical models developed in this dissertation make good use of a particular conjugate pair, the Dirichlet-multinomial, widely employed for discrete distributions.

The multinomial distribution

$$\text{Mul}(\boldsymbol{x}|n,\boldsymbol{\theta}) = \frac{n!}{\prod_{d=1}^{D} x_d!} \prod_{d=1}^{D} \theta_d^{x_d} \tag{2.3}$$

places mass on event counts $\boldsymbol{x} \in \mathbb{Z}^{*D}$ where $\sum_{d=1}^{D} x_d = n$. Each component x_i can be viewed as the number of occurrences of event i in n observations with D possible events. The parameter $\boldsymbol{\theta} \in \Delta^{D-1}$ is a point on the unit $(D-1)$-simplex, i.e., is a vector of D probabilities that sums to one. The notation for single-event multinomial draws is relaxed for convenience: a random variable $x \sim \text{Mul}(\boldsymbol{\theta})$ is the index of the positive entry in $\boldsymbol{x} \sim \text{Mul}(1,\boldsymbol{\theta})$.

The Dirichlet distribution, its conjugate prior, is a density over the unit simplex defined as

$$\text{Dir}(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \frac{\Gamma(\sum_{d=1}^{D} \alpha_d)}{\Pi_{d=1}^{D} \Gamma(\alpha_d)} \prod_{d=1}^{D} \theta_d^{\alpha_d - 1} \tag{2.4}$$

for vectors $\boldsymbol{\theta} \in \Delta^{D-1}$, where $\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$ is the gamma function and $\boldsymbol{\alpha} \in \mathbb{R}^{+D}$ controls the shape of the distribution. Because the support of the Dirichlet distribution ranges only over l_1-normalized vectors, it can be interpreted as a distribution over parameterizations of the multinomial distribution. Its conjugacy with that distribution makes it particularly convenient, and widely used.

This conjugacy can be interpreted in surprisingly intuitive ways. Formally, the posterior distribution over the parameters of an unknown multinomial distribution with a Dirichlet prior $\text{Dir}(\boldsymbol{\alpha})$, is given, after observing count vector $\boldsymbol{x}$, by $\text{Dir}(\boldsymbol{\alpha} + \boldsymbol{x})$. Less formally, draws from that unknown multinomial distribution can be thought of as draws in the equivalent Pólya-Eggenberger urn scheme (Elkan, 2006). Imagine an urn from which colored balls are drawn repeatedly with replacement. After returning each ball, another ball of the same color is added to the urn, increasing the number of balls inside. If the urn initially contains many balls, the additional balls have little effect, but if the urn begins nearly empty, they make colors already drawn more likely be drawn again. The starting configuration of the urn corresponds to the Dirichlet parameter vector $\boldsymbol{\alpha}$—which need not consist of integers.

2.7 Conclusions

This dissertation leverages the techniques of probabilistic reasoning to amplify the impressive efficiency gains produced by research on low-level automated reasoning over the past two decades. It therefore spans two worlds: the discrete world of propositional logic, and the continuous world of statistical inference. This chapter has provided a brief review of the relevant aspects of both worlds, starting with a high-level review of the solver performance landscape and ending with the details of the discrete distributions that Chapter 4 will employ. The next chapter moves beyond this background material to describe existing portfolio systems, the architectures they champion, and their weaknesses addressed by this dissertation.

Chapter 3

Related Work

The phenomenon of competitive, complementary solvers can be found in many different domains. Correspondingly, many algorithm portfolio systems have been reported in the literature. The number of unique architectures employed by these systems, however, is small. This chapter will review these architectures, their representative portfolio systems, and the ways in which this dissertation differs from and improves on them. It will also briefly review the research areas, such as hierarchical Bayesian modeling, underlying the methods applied by MAPP.

3.1 Economics and Naïve Parallel Portfolios

Consider the simplest possible portfolio: a suite of algorithms executed in parallel. Such a portfolio will behave the same on every instance. Changing one algorithm's share of processor time can, however, change the expected cost of obtaining a solution. One way to view such a suite of solvers is thus as analogous to a collection of financial assets, each providing a different return on investment. The owner of an *algorithm* portfolio invests computational instead of monetary resources, and it is from this perspective that Huberman et al. (1997) introduced the term "algorithm portfolio".

Important properties of both financial and algorithm portfolios include risk, defined as the standard deviation of the reward distribution, and expected reward or expected cost. Portfolios on the Pareto frontier dominate all other portfolios: for every other possible weighting of constituent algorithms, some portfolio on the frontier has greater expected reward, lower risk, or both. The economics approach therefore provides a framework for examining two key aspects of algorithm behavior, and effectively connects portfolio methods to the useful language of finance (Markowitz, 1952). Empirical data collected by Huberman et al. (1997) on the run time distribution of the Brélaz heuristic (Brélaz, 1979), a graph coloring heuristic also employed in general constraint-satisfaction methods (Smith, 1999), showed that these concepts could be put into practice in a standard domain.

Gomes and Selman (1997a) applied the economics framework to SAT, using empirical data from heuristics running on CNF encodings of the quasigroup completion problem (QCP), a synthetic benchmark (Gomes and Selman, 1997b). They also considered the general case of an arbitrary-size portfolio, and demonstrated the dramatic improvements to expected run time achievable by portfolios over individual solvers, even by portfolios assembled from multiple copies of the same algorithm. In that case, performance improves because multiple initializations mitigate the influence of heavy tails in run time distributions (Gomes et al., 1998).

One recent example of an effective naïve portfolio, `ppfolio`, entered the 2011 SAT Competition. It runs, in parallel, a set of solvers consisting largely of past competition winners (Roussel, 2011). It served as a simple but effective baseline portfolio, and, surprisingly, won more medals in the final accounting than any other single system (Järvisalo et al., 2011b). It will be included in the head-to-head portfolio comparisons conducted in Chapter 7.

The simplicity of the economics approach makes it a useful intellectual contribution but limits its generality. For example, it typically penalizes upside and downside risk equally; it assumes that run time distributions are known; it does not consider the role of

information; and it does not permit portfolios to be adjusted once their execution begins. Despite its early influence, most modern algorithm portfolios do not employ parallel execution or the economics perspective. Instead, they focus on selecting a single algorithm for a given instance.

3.2 Algorithm Selection

The most straightforward way to employ a suite of solvers is to pick one of them to run—and then run it. From this perspective, the problem is clear but difficult: given only superficial information about an instance, select the algorithm that will perform best. Most successful portfolio methods attempt to solve this problem directly, and most use a standard supervised learning approach to do so. This section broadly reviews their contributions.

3.2.1 The Selection Problem

The choice of best-suited algorithm and configuration for a problem instance was formalized as "algorithm selection" by Rice (1976, 1979). His work identified essential elements of the problem: identifying performance criteria, evaluating algorithm behavior, extracting superficial features of problem instances, and using those features to choose an algorithm.

The line of research directly inheriting from this early work has focused on advice-giving systems to aid practitioners in scientific computation (Weerawarana et al., 1996; Houstis et al., 2000), and on the selection of partial differential equation (PDE) solvers in particular. This area has remained largely separate from the SAT-centered field of portfolio methods, which focuses on *automatic* selection based on instance features. The independent development of such systems for PDEs, however, provides further evidence that the sensitivity of heuristic methods to parameterization and algorithm selection is a concern that extends far beyond SAT. The next subsection presents automatic feature-based selection methods in depth.

3.2.2 Supervised Selection and `SATzilla`

`SATzilla` (Nudelman et al., 2004; Xu et al., 2008a, 2009) is the most prominent modern portfolio method, at least in SAT, partly due to its dominant performance in the 2007 and 2009 solver competitions (Le Berre et al., 2007, 2009). It is the chief representative of feature-oriented, supervised algorithm-selection methods, which focus on the two most significant of the research problems identified by Rice (1976): extracting information about problem instances, and exploiting that information effectively.

These methods build on work that examines how features of an instance, i.e., efficiently-computable functions of a problem representation such as CNF, offer clues to the difficulty posed by that instance to particular solvers. Such "empirical hardness models" have been motivated by dramatic results in work on *generating* difficult instances, and especially by the striking phase transition in the landscape of random SAT: as random instances become more constrained, as parameterized by the ratio of clauses to variables, they undergo a rapid shift from satisfiable to unsatisfiable and the computational cost required to solve them spikes (Mitchell et al., 1996; Monasson et al., 1999). For the standard random k-SAT instance distribution, in other words, the clauses-to-variables ratio is particularly predictive of solver run time.

Early evidence specifically in hardness prediction includes the feature survey of Nudelman et al. (2004), the earlier survey and prediction experiments of Horvitz et al. (2001), and the analysis of combinatorial auctions by Leyton-Brown et al. (2002). Among recent demonstrations of these ideas are successes in instance-based solver parameterization (Nikolić et al., 2009) and problem-sensitive restarting using progress statistics (Sinz and Iser, 2009). A vast amount of information about a SAT instance can be captured by these features. In general, they indicate the overall constrainedness of clauses in the expression, the overall balance of variables and clauses, and the behavior of length-limited runs of certain search procedures. Table 3.1 summarizes the features used by the 2007 version of `SATzilla` (Xu et al., 2008a).

Group	Count	Feature
Problem size	1	Number of clauses
	1	Number of variables
	1	Clauses-to-variables ratio
Variable-clause graph	5	Variable node degree statistics
	5	Clause node degree statistics
Variable graph	4	Node degree statistics
Balance	3	Ratio of per-clause $+/-$ literals
	5	Ratio of $+/-$ occurrences of each literal
	2	Fraction of binary and ternary clauses
Horm proximity	1	Fraction of Horn clauses
	5	Number of variables in Horn clauses
DPLL probing	5	Number of unit propagations
	2	Estimated search space size
Local search probing	4	Number of steps to the best minimum
	1	Average improvement to best in a run
	2	Gain due to first local minimum
	1	Variance in unsat. clauses at each min.

Table 3.1: Summary of features used in the 2007 version of `SATzilla` (Xu et al., 2008a), organized into groups by kind. In all, 48 features are used in the 2007 version, while the 2009 version uses 84 features on large industrial instances and 96 features on all others (Xu et al., 2009). Many features can be computed, and their predictive power is primarily responsible for the success of feature-oriented portfolios like `SATzilla`.

In an algorithm-selection portfolio, supervised learning is used to build per-solver predictors of properties of behavior, such as the time taken by the solver to solve an instance. Training instances are used to build examples: the feature vector computed for each instance is labeled with a score of the solver's performance on that instance. In many cases, that score may simply be the duration of the solver's run on that instance. Since solvers are not guaranteed to terminate in a reasonable amount of time on every problem instance, however, they are killed after some cutoff, and the supervised learning method must cope with this sample censorship. In other cases, some other measure of performance is targeted, such as a weighted combination of efficiency and success probability (Xu et al., 2008b).

More complex supervised learning schemes can also improve prediction. Xu et al. (2007), for example, built separate models for satisfiable and unsatisfiable instances, as well as for the satisfiability of an instance. The latter predictor was used to merge the two class-specific predictions at test time. Gebser et al. (2011b) found this mixture-of-experts approach to be less helpful in ASP, however.

On each test instance, the computed feature vector is given to each learned predictor, and the portfolio runs the solver expected to perform best. This architecture is diagrammed in Fig. 3.1. `SATzilla` makes this overall procedure more complex in three ways: by identifying a backup solver and a set of pre-solvers during training, by briefly running pre-solvers prior to feature computation, and by predicting the cost of feature computation—skipping it and running the backup solver if the computation is likely to be too expensive (Xu et al., 2008a, 2009). Other algorithm-selection portfolios, described next, adapt the standard architecture in other ways.

3.2.3 Other Supervised Portfolios

The standard supervised algorithm-selection architecture championed by `SATzilla` has been altered, adapted, and tuned in many ways for many different domains. This section briefly surveys these associated methods. In general, however, the core of the architecture

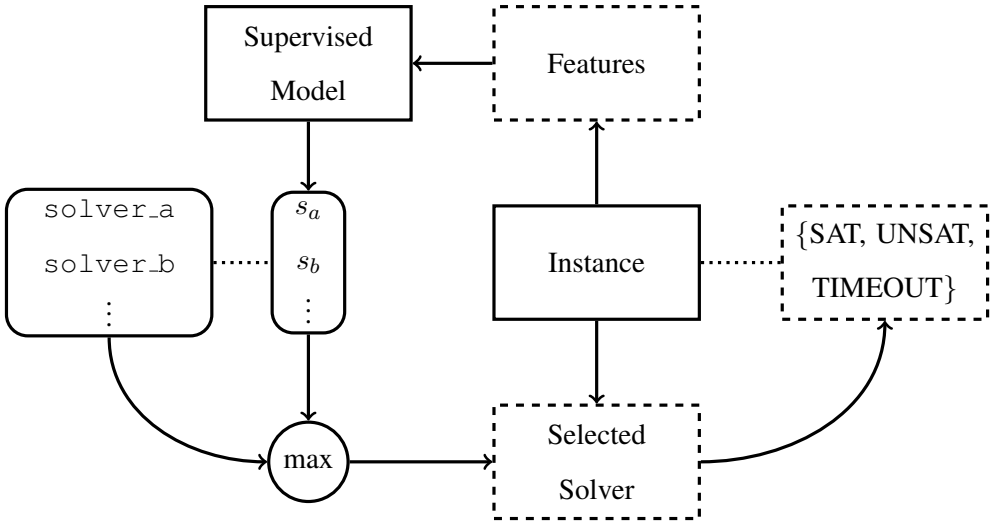


Figure 3.1: The operation of a supervised-selection portfolio, such as SATzilla. Arrows indicate the flow of information and control. Features are computed from an instance; from those features, a previously-trained model predicts the future performance score s of each solver on that instance; and the solver with the highest predicted score is selected and run until completion or timeout. This architecture is straightforward and effective, but restricted to the selection of single solvers.

remains the same: supervised learning is used to select an algorithm to run.

Nikolić et al. (2011) employed a k-nearest neighbor (k-NN) classifier for algorithm selection in SAT. Surprisingly few portfolios have been developed for close relatives of SAT; the MAX-SAT portfolio of Matos et al. (2008) is one of those few. Like SATzilla, it used ridge regression, but first applied forward feature selection and basis-function feature expansion.

Gebser et al. (2011b) built an algorithm selection portfolio, claspfolio, for ASP. Support vector regression (SVR) was used both with static instance features and with dynamic search features collected from multiple restarts of the clasp ASP solver. Their solver suite was comprised of multiple configurations of clasp. Experiments in

Section 8.3 will include `claspfolio`.

Pulina and Tacchella (2007, 2009) designed a portfolio for QBF and tested several different classifiers, including decision trees, decision rules, logistic regression, and nearest-neighbor, on solvers and benchmarks in that domain. In an interesting extension of the architecture, they moved beyond offline training to consider online adaptation. Their portfolio tried multiple solvers on test instances, and updated its model based on the additional run time information it acquired.

Some algorithm-selection methods have targeted specific applications. Arbelaez et al. (2010) constructed a portfolio of constraint optimization algorithms for protein structure prediction. It employed instance features that analyzed the original protein structure problem, rather than one of its encodings. They evaluated both decision trees and linear regression as their supervised learning method, finding advantages to each. Messelis and De Causmaecker (2011) applied supervised algorithm selection to the computation of hospital nurse rosters, using application-specific features that characterize scheduling constraints (Messelis and De Causmaecker, 2010). Such examples illustrate the breadth of the need addressed by portfolio methods.

3.2.4 Non-Model-Based Portfolios

Supervised algorithm-selection portfolios now consistently dominate individual solvers in competition. They convincingly demonstrate the potential of portfolio methods. This basic portfolio architecture, however, suffers from two limitations. First, predictions are individual solver performance scores. While such scores can incorporate tradeoffs between multiple selection criteria, it is unclear how they can be generalized to arbitrary distributions over run times and outcomes. Second, as in the early work of Rice, decisions are restricted to a small fixed set. `SATzilla`, for example, makes only two choices: whether to compute features, and which single solver to execute thereafter. In effect, the `SATzilla` method builds a set of static portfolio schedules, one for each solver, and chooses between those

schedules at run time. Other possible schedules, even the simple schedules employed by naïve parallel portfolios, cannot be chosen, even though their performance can be superior.

The probabilistic portfolio work presented in this dissertation is influenced by these selection-based portfolios, and it employs the informative instance features that they have identified. However, this dissertation emphasizes the prediction of complete solver performance distributions instead of aggregate scores, and it advocates the use of general execution schedules that include solver restarts.

One important deviation from the regression approach of solvers like `SATzilla` is implemented by a portfolio named `3S`, the "Satisfiability Solver Selector", which entered the 2011 SAT competition. It finds an instance's nearest neighbors in feature space, then follows a solver execution schedule tailored to those instances (Kadioglu et al., 2011; Malitsky et al., 2011). In the field of constraint programming (CP), O'Mahony et al. (2008) built a similar algorithm portfolio named `CPHydra`. It used "case-based reasoning"—essentially k-NN—and also computed a custom schedule.

One drawback of the k-NN approach is that it places as much weight on features irrelevant to solver performance as it does on features that are informative. Kroer and Malitsky (2011) address this shortcoming in the related "ISAC" system, which clusters instances in feature space and tunes solvers to particular clusters (Kadioglu et al., 2010), using clever feature selection. In contrast, the feature integration approach of MAPP, described in Chapter 5, naturally handles this problem.

The use of tailored execution schedules in `3S` and `CPHydra` makes them closer in spirit to the approach advocated in this dissertation. Their focus remains, however, on deterministic solvers, and their architecture, methods, and focus are correspondingly different from those employed in this dissertation. Furthermore, Chapter 7 will find that MAPP can often outperform `3S`, especially on domains that favor highly nondeterministic SLS solvers.

3.3 Solver Execution Scheduling

Solver scheduling, like that done by 3S and MAPP, is ignored in the algorithm selection perspective but is nonetheless fundamental to portfolio operation. It has long been recognized as an important aspect of performance in nondeterministic algorithms. The seminal work of Luby et al. (1993), for example, derived the optimality of a "universal" cutoff schedule in the case of total uncertainty: an exponential schedule of the form $\langle 1, 1, 2, 1, 1, 2, 4, \ldots \rangle$. Modern SAT solvers often schedule internal search restarts according to this scheme (Huang, 2007).

Solvers, however, are not scheduled under total uncertainty. Some information about their run time distributions is often available: solvers may have been run on similar, previously-observed instances, for example. Streeter et al. (2007a) introduced greedy algorithms for the offline, learning-theoretic, and online settings of this solver-scheduling problem, and gave theoretical error bounds for each. Streeter et al. (2007b) applied these ideas to solver restart scheduling, and Streeter and Smith (2007) showed how to extend them to optimization problems.

Chapter 6 will introduce a planning approach that tackles the offline problem, but employs a dynamic programming algorithm that, in some cases, can guarantee an optimal execution schedule. It will be compared against the greedy approach. O'Mahony et al. (2008) present a similar solver-scheduling objective for their work in CP, but employ inefficient direct search to solve it for a limited number of solvers and instances. They make a connection to the knapsack problem to argue for the intractability of tackling their objective directly—and to identify solver scheduling as an open problem. This dissertation makes a similar connection, but uses it instead to develop an efficient scheduling algorithm.

Other approaches to solver scheduling include that of Kadioglu et al. (2011), who employed general integer programming (IP) to compute portfolio schedules that do not include solver restarts, and that of Ruan et al. (2002), who treated single-solver restart scheduling under partial uncertainty as a Markov decision process (MDP), applying dy-

namic programming to solve it exactly but inefficiently. Neither of these approaches addresses solver scheduling in the same general sense, or as efficiently, as the method presented in this dissertation.

3.4 Bandit Portfolios

One very different body of research has explored the connection between algorithm portfolios and so-called "bandit" algorithms, strategies for sequential decision making that provide formal guarantees. Bandit algorithms specifically consider the case of actions repeatedly selected from a fixed set of k choices, i.e., repeated pulls on a k-armed bandit, when the arms provide rewards drawn either from a fixed distribution (the stochastic case, i.e., a one-state MDP) or arbitrarily (the nonstochastic case). They typically guarantee bounded *regret* against a particular strategy, repeated pulls of the arm that provides the highest mean reward, in hindsight (Lai, 1987; Auer et al., 2002b,a).

Bandit algorithms are general and useful. It is difficult, however, to apply a standard bandit method to the portfolio setting because actions in that setting, i.e., solver runs, do not typically provide individual reward. Instead, reward is obtained only once, when an instance is solved, at which point the trial ends. A small body of work reformulates the bandit setting to apply more directly to portfolios. In this *max bandit* setting, the goal is to maximize the expected maximum payoff over trials. Unlike the nonstochastic bandits literature applied elsewhere, payoffs from a bandit arm are assumed to be drawn independently from a fixed distribution. In the work of Cicirello and Smith (2005) and Streeter and Smith (2006a), this distribution is assumed to be in the extreme value family; this assumption is relaxed by Streeter and Smith (2006b).

The max bandit work is instructive in its contrasts with the traditional bandit literature. It does not, however, exploit knowledge of solver behavior gained from running on previously-observed problem instances. Furthermore, it applies only when solvers yield multiple suboptimal but non-zero payoffs on individual instances, as in optimization. For

both of these reasons, it is unattractive as a general approach to portfolio decision making.

Another way that bandits can be applied to the portfolio setting is by using a coarse-grained action set, choosing complete solver execution schedules instead of individual solver runs. Streeter et al. (2007a), for example, used methods from the label-efficient bandits literature (Cesa-Bianchi et al., 2004) to bound scheduling regret in the online setting. In an elegant extension, Streeter and Smith (2008) applied a sleeping-experts bandit algorithm (Blum and Mansour, 2007) to improve scheduling on problem instances labeled with binary features. A schedule is constructed over those instances manifesting each feature. On new instances, the bandit algorithm then selects only from the schedules that apply. Regret is thus bounded simultaneously against all applicable schedules.

Other work in this vein has focused on parallel schedules. Gagliolo and Schmidhuber (2006) developed heuristic time allocators and used an appropriate bandit method (Auer et al., 1995) to weight them according to their performance. Their most elaborate allocator employed the nearest-neighbor style estimator of Wichert and Wilke (2005), one of which is learned for each solver; others included the baseline naïve parallel allocator. This framework was applied to the restarts problem by Gagliolo and Schmidhuber (2007), and was extended to the case of an unknown bound on the loss, the cost of solving an instance, by Gagliolo and Schmidhuber (2011).

In general, such bandit results focus on the online setting: they attempt to provide performance guarantees throughout the learning process. This dissertation focuses instead on the offline setting. This focus permits more complex models and stronger statistical assumptions about instances, and can thus potentially infer more information from solver run data.

3.5 Domain-Focused Probabilistic Models

The use of model-based inference to predict the values of unobserved variables is central to statistics. Complex, hierarchical, generative models of discrete data, however, have gained

popularity only as the sinking cost of computation has made sampling and inference for these models practical. This section reviews the use of models in predicting solver performance specifically, and the success story of generative models, particularly in text, more generally.

3.5.1 Models of Solver Performance

At a high level, portfolio methods acquire information, update their beliefs based on that information, then act on those beliefs. In the Bayesian view, applied to solver behavior by Horvitz et al. (2001) and Kautz et al. (2002), a probabilistic model represents an agent's assumptions and inferred probabilities represent its beliefs. Their work has had substantial influence on the probabilistic portfolio framework outlined in this dissertation.

Three important sources of information were identified: contextual evidence about the distribution of problem instances, structural evidence about the superficial features of an instance, and execution evidence about the visible aspects of solver behavior. Experiments conducted by Horvitz et al. (2001) ran two solvers on QCP instances, collecting evidence over an initial observation horizon. The duration of each solver run was labeled "short" or "long", and a Bayesian learner—one that assumes a decision tree form for the conditional distributions—built a model of that labeling. Much of the work of Horvitz et al. (2001) examined the predictive features in that model, but it also made simple restart/continue decisions for solver runs. Kautz et al. (2002) further developed that application.

This Bayesian perspective is attractive because it is general: it incorporates diverse sources of evidence, it can model arbitrary aspects of solver behavior, and its probabilistic predictions are often easy to incorporate into decision making. The specific system developed in this prior work was, however, limited. It modeled only individual solvers, assumed a fixed, potentially-expensive observation window, and predicted only the short/long class label. This dissertation shares the Bayesian perspective, but adopts a new set of techniques and combines them into a system for the more general algorithm portfolio problem.

3.5.2 Models of Natural Language Text

Statistical models of bag-of-words (BOW) data have received substantial interest over the past decade. So-called "topic models" in particular, which seek to infer semantically coherent word clusters underlying natural language text corpora (Blei et al., 2003), have seen numerous extensions and applications: to image data (Fei-Fei and Perona, 2005; Li et al., 2011), to multi-language corpora (Mimno et al., 2009), to the measurement of scientific impact (Gerrish and Blei, 2010), to voting records in the United States Senate (McCallum et al., 2007), even to archaeology (Mimno, 2009). While the models of Chapter 4 are motivated directly by solver performance prediction, and do not build on any specific existing model in a different domain, the statistical tools of mixture models and Dirichlet-multinomial conjugacy are shared across all of these applications.

Unlike the work of Horvitz et al. (2001), which focused on modeling the network of statistical interactions among static instance features, dynamic progress information, and solver performance, the modeling work in this dissertation is focused on inferring solver run time better based only on patterns inferred in those data. Given a small number of runs of each solver, the goal is to predict the statistics of future runs. From one perspective, models of text like that of Blei et al. (2003) are engaged in a similar activity akin to imputation: inferring the statistics of a given document, were one to somehow extend that document to be infinitely long.

The most similar text models are those of "bursty" term distributions, which use the properties of Dirichlet-multinomial conjugacy discussed in Section 2.6—highlighted by the Pólya urn interpretation—to handle the situation in which a term appears either repeatedly or not at all (Elkan, 2006; Katz, 1996; Madsen et al., 2005; Doyle and Elkan, 2009). This phenomenon is widespread. One example is sports writing: although team names are semantically similar, a given article contains the names of only a few. Those few teams, however, are mentioned repeatedly; once a name is seen once, it is likely to be seen again. The same phenomenon can occur in solver behavior. A deterministic solver,

for example, may have some distribution over performance if the instance is unknown, but, in repeated runs on that instance, the same run time will be repeatedly drawn. In the portfolio domain, Silverthorn and Miikkulainen (2010) introduced a pair of discrete latent class models, focused explicitly on modeling solver burstiness, that handle arbitrary solver outcomes. Chapter 4 will develop simpler, more effective descendants of those models by specializing in decision problems.

3.6 Conclusions

The portfolio architecture developed and evaluated by this dissertation stands apart from the algorithm-selection architecture employed by other successful portfolio methods. It advocates a focus on probabilistic algorithms and inference of run time distributions, neither of which play a significant role in other approaches. It computes general execution schedules, something rare in other approaches, and introduces a simple and efficient approach to doing so. Its integration of instance features brings together the strengths of the model-based and nonparametric approaches, allowing the mature and effective methods of supervised learning to provide information about complete distributions. Taken as a whole, this modular architecture contributes a novel perspective to the field of algorithm portfolio design.

Chapter 4

Modeling Solver Performance

An algorithm portfolio must choose solvers to execute on a new instance without knowing, in advance, exactly how those solvers will perform. Training data provide information about typical solver performance on the distribution of instances at hand, but this information is incomplete and limited to past experience. The portfolio is therefore neither entirely ignorant nor entirely informed; it must cope with uncertainty. The fundamental principle of the MAPP approach, as outlined in Section 1.4, is the representation of a portfolio's uncertainty as a weighted set of possible scenarios. Each member of this set is an estimate of the solver suite's performance on a specific training instance. To compute those estimates, this chapter develops a family of probabilistic graphical models of solver performance, and compares their predictions on SAT solver performance data.

4.1 Run Time Distributions and Las Vegas Algorithms

Heuristic solvers are examples of so-called *Las Vegas* algorithms. While a Monte Carlo algorithm returns an approximate result in a predictable, deterministic amount of time, a Las Vegas algorithm returns an exact result in an unpredictable, nondeterministic amount of time (Babai, 1979). The distribution over the time taken by running a Las Vegas algorithm

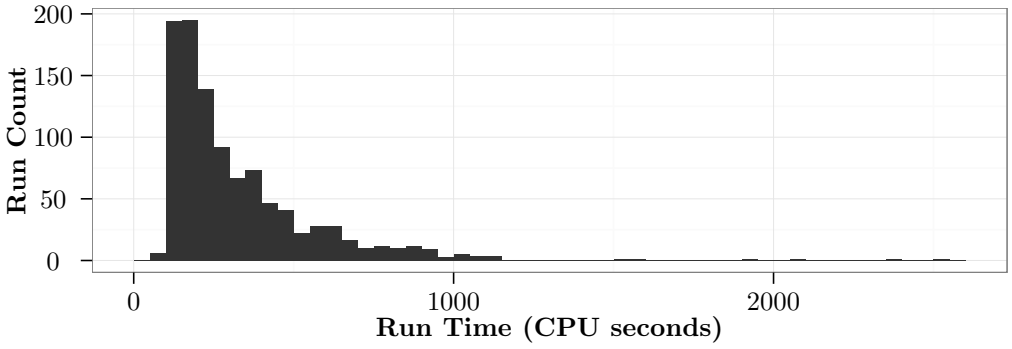


Figure 4.1: The run time distribution of the SAT solver `lingeling` (v276) executing on the instance `blocks-blocks-37-1.150-SAT.cnf` from the 2011 SAT competition, captured by visualizing 1024 runs as a histogram with a bin width of 50 s. While many runs succeed in little more than 100 s, some take upwards of 1,000 s. The performance of `lingeling` on this instance would be poorly represented by its expected run time or by assuming that it operates deterministically.

to completion on some instance is known as its run time distribution (RTD) on that instance.

RTDs have received less attention as a SAT research topic with the rise of CDCL solvers, in which randomness plays an arguably smaller role relative to SLS solvers. Modern CDCL solvers, however, remain far from deterministic. As an example, Figure 4.1 shows the RTD of a modern CDCL SAT solver, `lingeling`, running on a planning instance included in the most recent SAT competition.

Knowledge of solvers' RTDs on an instance enables a portfolio to schedule solvers intelligently and thus to maximize its probability of success. A portfolio, however, has no certain knowledge of how solvers perform on an instance it has not previously seen. Instead, a portfolio must assume that solvers' performance on a new instance is in some way related to their performance on training instances. This dissertation makes a strong, specific assumption along these lines: that solvers' RTDs on a new instance are identical to

their RTDs on some unknown training instance. Training instances, in other words, form a discrete mixture model of test instances. This modeling assumption is most appropriate when all instances are drawn from the same, fixed distribution, as when training instances are sampled from a larger corpus of unsolved instances.

A portfolio does not observe solvers' complete RTDs even on its training instances: it observes only individual runs on those instances, i.e., samples from RTDs. Furthermore, collecting many samples from each instance becomes impractical. Under the per-instance cutoff used for SAT competition instances, for example, a single run may consume nearly two hours of processor time. The challenge in training a probabilistic portfolio is therefore to estimate solvers' RTDs on training instances with sufficient accuracy given only a few of these expensive samples.

The following sections build a sequence of increasingly sophisticated models of solver runs. Each model builds on an earlier model, but captures a new aspect of solver behavior. As experiments will show, better models allow solver behavior to be captured more accurately from fewer runs.

4.2 General Methodology

Throughout this dissertation, solver run outcomes were reused across experiments by sampling them from a database collected on a local cluster of identical machines (Xeon X5355 @ 2.66 GHz) over tens of thousands of hours of processor time. This allowed models and portfolios to be compared without re-executing every solver in every experiment. The per-instance run time cutoff of 6,000 s was selected to encompass that used by the SAT competition. Whenever applicable, plots are shaded to indicate 95% confidence bounds. Finally, all source code and data are publicly available (Silverthorn, 2012).

A standard cross-validation scheme was used to to measure the applicability of each model empirically. A collection of problem instances was randomly split into training and test sets, the model was fit to the training data, and the probability assigned by the model to

the held-out test data was measured. A model can be considered a better fit if it considers the test data more likely. Throughout this chapter, unless otherwise specified, experiments employ data collected by running the suite of solvers used by `SATzilla2009` on every instance included in the 2007 SAT competition.

4.3 A Family of Run Time Distribution Models

The MAPP modeling approach will treat run time distributions as discrete, like the histogram in Fig. 4.1. Discrete models lump runs of sufficiently similar duration together in the same bin, and can thus be used only to predict the bin in which a run is most likely to fall. Discretization, however, simplifies both modeling and planning: efficient, general-purpose discrete distributions, such as the multinomial, can be simpler to estimate, and restricting reasoning to a finite number of actions makes it simpler to plan. Chapter 6 quantifies the cost of discretization. Furthermore, the multinomial distribution places no restriction itself on the shape of a solver's performance distribution.

Runs that fail, either because they exceed the cutoff or because they terminate abnormally, fall into an arbitrarily-designated failure bin. Let S be the number of solvers, B the number of modeled histogram buckets (with failing runs in the final bucket, $b = B$), I the number of problem instances, R_{si} the number of runs made by solver s on instance i, and b_{sir} the index of the bin into which run r of solver s on instance i falls. Each bin index is a random variable, one that depends both on a solver's complex deterministic behavior and on its internal randomness. The generative models developed in the following sections will be used to estimate the underlying RTDs from observed values of b_{sir} for many different s, i, and r.

4.3.1 Maximum-Likelihood Multinomials

The simplest generative process for the discrete data in this setting is one in which each bin index is drawn from an unconstrained multinomial distribution specific to each solver on

each instance, i.e.,

$$b_{sir} \sim \text{Mul}(\boldsymbol{\theta}_{si}), \qquad \begin{aligned} s &\in 1 \ldots S, \\ i &\in 1 \ldots I, \\ r &\in 1 \ldots R_{si}, \end{aligned} \tag{4.1}$$

where each $\boldsymbol{\theta}_{si}$ is a parameter vector of a multinomial RTD, and each element θ_{sij} is therefore the probability $p(b_{sir} = j)$ that run r of solver s on instance i will fall into bin j.

In general, of course, these probabilities are unknown. Instead, various b_{sir} values are observed in training, and the latent $\boldsymbol{\theta}_{si}$ parameters inferred. The maximum likelihood (ML) parameters are those that place maximum probability mass on the observed data. In this setting, the ML parameters are simply the observed rates of runs falling into each bin:

$$\theta_{sic} = \frac{1}{R_{si}} \sum_{r=1}^{R_{si}} \delta_{cb_{sir}}. \tag{4.2}$$

As more data become available, the observed rates converge to the true rates and the ML parameters converge to the true parameters. This convergence in the limit, however, is a poor guarantee, since each additional sample may require hours of processor time to obtain.

4.3.2 Multinomials with Uniform Smoothing

The ML multinomial parameters place no probability mass on unseen events, which becomes problematic when few samples are available from any solver on a training instance. If only one sample is available, for example, then the maximum-likelihood multinomial distribution will predict with certainty that every future run will be identical to the observed run. That prediction is reasonable only if the solver is deterministic and the machine entirely unloaded.

Addressing this shortcoming involves moving probability mass from observed outcomes to unobserved outcomes. Here, mass is shifted by incorporating a prior belief that the parameter vector of each multinomial RTD is drawn from a smooth distribution over the

unit simplex. The Dirichlet distribution is a convenient, conjugate prior for this purpose, leading to the hierarchical generative process

$$\boldsymbol{\theta}_{si} \sim \text{Dir}(a\mathbf{1}), \qquad s \in 1 \ldots S, \quad i \in 1 \ldots I, \tag{4.3}$$

$$b_{sir} \sim \text{Mul}(\boldsymbol{\theta}_{si}), \qquad s \in 1 \ldots S, \quad i \in 1 \ldots I, \quad r \in 1 \ldots R_{si}, \tag{4.4}$$

where $\mathbf{1}$ is a B-dimensional vector of ones and a a scalar hyperparameter that controls the amount of smoothing. Section 4.4 discusses how to obtain information about a model's posterior distribution over RTDs, given observed runs.

Figure 4.2 plots the probability assigned to test data under various levels of uniform smoothing. It makes clear the importance of placing mass on unobserved events, and it establishes a baseline model. The obvious shortcomings of uniform smoothing motivate the development of more complex models that implement more informed priors over the RTDs.

4.3.3 Multinomials with Non-Uniform Smoothing

The approach of Section 4.3.2 spreads a fraction of the available probability mass uniformly across all possible outcomes. Furthermore, it smooths all solvers equally. If, for example, a given solver is particularly bad (i.e., its runs most often fall into the failure bin), or particularly good (i.e., its runs most often fall into a bin associated with quick success), predictions of that solver's RTD may benefit from a prior belief that places mass in those particular bins.

This flexibility can be expressed by employing, instead of a single symmetric Dirichlet prior for all solvers, a separate asymmetric Dirichlet prior for each solver. The full

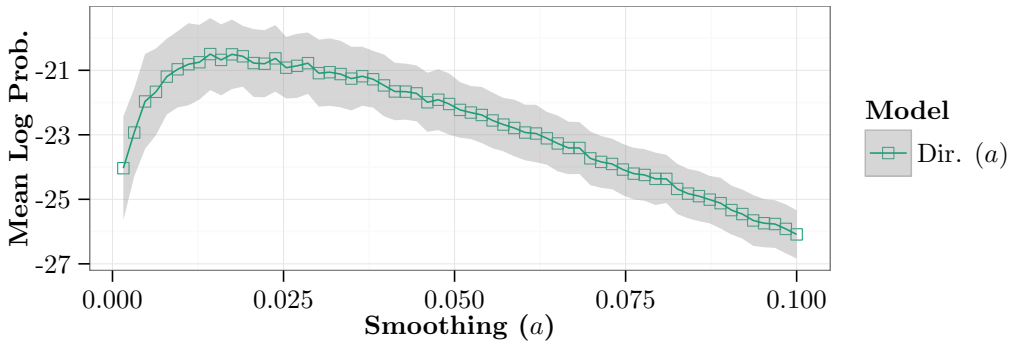


Figure 4.2: The mean log probability of test data, using ten-fold cross validation, under the Section 4.3.2 smoothed-multinomial model and varying values of the uniform-smoothing parameter a. With minimal smoothing, too little probability mass is shifted to events present in the test data but unobserved in the training data. With substantial smoothing, too much probability mass is shifted to events unobserved in training that truly never occur. Later models will smooth more selectively, attempting to shift mass primarily to unobserved events considered likely to occur.

generative process for this model becomes

$$\boldsymbol{\theta}_{si} \sim \text{Dir}(\boldsymbol{\alpha}_s), \qquad s \in 1 \ldots S, \quad i \in 1 \ldots I, \tag{4.5}$$

$$b_{sir} \sim \text{Mul}(\boldsymbol{\theta}_{si}), \qquad s \in 1 \ldots S, \quad i \in 1 \ldots I, \quad r \in 1 \ldots R_{si}, \tag{4.6}$$

where each of the Dirichlet hyperparameters α_{sb} quantifies a prior expectation that a run of solver s will fall in bin b on a new instance.

4.3.4 Multinomials with Independent Mixture Smoothing

Asymmetric, solver-specific smoothing captures an important aspect of the data. Going further involves leveraging another pattern in solver performance. It is clear that each problem instance is not unique: the distribution of instances may, for example, include both "easy" instances, on which a solver may succeed quickly, and "difficult" instances, on which it may struggle. In other words, prior beliefs may not be well represented by any single Dirichlet parameter vector, even a solver-specific vector, but rather by a mixture of such vectors.

This type of prior belief can be captured by per-solver finite mixtures of Dirichlet distributions. For each solver, a run is drawn from a multinomial distribution whose parameters are in turn drawn from one member of a set of K Dirichlet distributions. Specifically:

$$z_{si} \sim \text{Mul}(\boldsymbol{\beta}_s), \qquad s \in 1 \ldots S, \quad i \in 1 \ldots I, \tag{4.7}$$

$$\boldsymbol{\theta}_{si} \sim \text{Dir}(\boldsymbol{\alpha}_{sz_{si}}), \qquad s \in 1 \ldots S, \quad i \in 1 \ldots I, \tag{4.8}$$

$$b_{sir} \sim \text{Mul}(\boldsymbol{\theta}_{si}), \qquad s \in 1 \ldots S, \quad i \in 1 \ldots I, \quad r \in 1 \ldots R_{si}. \tag{4.9}$$

Each z_{si} is the index of the solver-specific *latent class*, or *component*, to which instance i belongs, and each $\boldsymbol{\beta}_s$ is a K-dimensional parameter vector specifying the mixture proportions of those classes in the training data.

When a distribution of instances contains multiple groups, such as "easy" and "hard", the solver-wide priors of Section 4.3.3 capture any one group of instances poorly. In the same situation, the mixture model of Eqs. (4.7) to (4.9) can employ a different prior for each group. Even a single run may be strong evidence that an instance belongs to a particular group, and an inferred RTD can therefore reflect information captured in runs of the same solver on other instances.

4.3.5 Multinomials with Linked Mixture Smoothing

More generally, latent classes may be related across solvers. An instance that is difficult or easy for one solver may be difficult or easy for another; or, for example, a distribution may contain both satisfiable and unsatisfiable instances, with each kind best suited for a certain type of solver. Relationships between solvers' latent classes can be modeled by unifying the latent class indices for each instance, as in

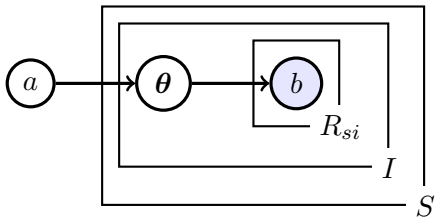
$$z_i \sim \text{Mul}(\boldsymbol{\beta}), \qquad i \in 1 \dots I, \tag{4.10}$$

$$\boldsymbol{\theta}_{si} \sim \text{Dir}(\boldsymbol{\alpha}_{sz_i}), \qquad s \in 1 \dots S, \quad i \in 1 \dots I, \tag{4.11}$$

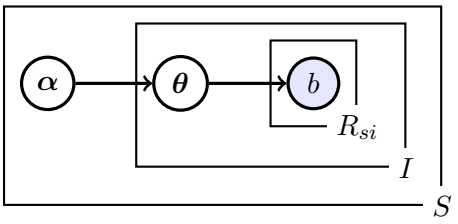
$$b_{sir} \sim \text{Mul}(\boldsymbol{\theta}_{si}), \qquad s \in 1 \dots S, \quad i \in 1 \dots I, \quad r \in 1 \dots R_{si}. \tag{4.12}$$

Rather than an instance belonging to multiple latent classes, one for each solver, each involving a single Dirichlet distribution, it now instead belongs to a single latent class that involves multiple Dirichlet distributions, one for each solver. This change means that the model may shift probability mass among bins for one solver based on a run made by a *different* solver on the same instance.

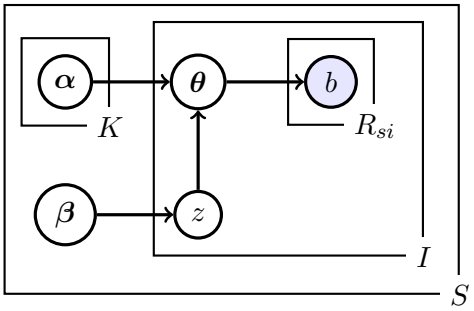
The four models developed in Sections 4.3.1 to 4.3.5 are visualized through the plate diagrams in Fig. 4.3. An important empirical question is whether the additional structure in Sections 4.3.4 to 4.3.5 improves the models' predictive power. Figure 4.4 plots the probability of test data under the mixture models, varying the number of mixture components K. The increase in probability with K suggests that these mixture priors do, in fact, capture an important aspect of solver run time data.



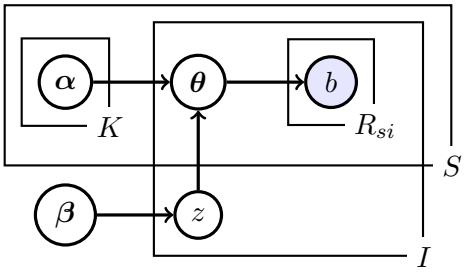
(a) The model of Section 4.3.2. All RTDs are smoothed uniformly and equally.



(b) The model of Section 4.3.3. Each solver's RTDs are smoothed separately and non-uniformly.



(c) The model of Section 4.3.4. Each solver's RTDs are smoothed non-uniformly by a separate mixture of latent classes.



(d) The model of Section 4.3.5. Each solver's RTDs are smoothed non-uniformly by a shared mixture of latent classes.

Figure 4.3: Sequence of plate diagrams illustrating the models from Sections 4.3.2 to 4.3.5. Shaded nodes represent observations, unshaded nodes latent variables, and arrows conditional dependencies. Boxes duplicate their contents according to their labeled counts. All models represent RTDs as multinomial distributions, but differ in how those distributions weight unobserved events. More accurate inference of unobserved-event probabilities should make it possible to compute better solver execution schedules.

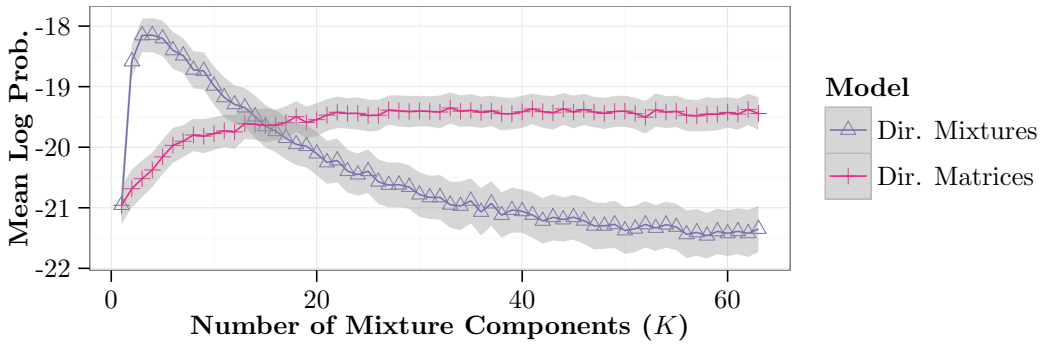


Figure 4.4: The mean log probability of held-out test data under the mixture-prior models of Sections 4.3.4 to 4.3.5, 64 random 80% train/test splits, and varying the number of mixture components K. While independent mixtures are unconstrained by those of other solvers, and fewer mixture components are therefore required to fit the data, they also overfit more easily. In general, the improvement in probability with K suggests that the mixture priors are useful additions to these models.

4.4 Model Inference and Application

Conditioned on the observed durations of solver runs on training instances, a particular model implies a particular posterior distribution over the multinomial parameters representing solvers' RTDs on those instances. Since RTDs are estimated offline, during portfolio training, the efficiency of model inference is not a primary concern—especially when compared to the immense computational cost of obtaining solver run data. This fact expands the set of models and inference methods that can be applied. Here, as in earlier work (Silverthorn and Miikkulainen, 2010), an EM algorithm is used to estimate ML parameter settings of the Dirichlet hyperparameters, which are then optimized using the fixed-point procedures of Wallach (2008) and Minka (2009).

The planning methods presented in Chapter 6 can incorporate model uncertainty

into their reasoning. Instead of computing solvers' mean RTDs for each instance, multiple multinomial settings can therefore be returned from the estimation procedure. For each instance under the linked mixture model, this procedure computes the maximum a posteriori (MAP) set of RTDs for each mixture component, weighting them according to the observed runs and estimated $\boldsymbol{\beta}$. For each instance under the independent mixture model, this procedure obtains possible sets of RTDs by repeatedly sampling z_{si} indices based on the estimated values of the other variables, then computing MAP RTDs corresponding to the sampled indices.

These sets of estimated multinomial parameters form a finite mixture model of solvers' RTDs on a particular distribution of problem instances. The next section will compare the predictive utility on SAT competition instances of estimates obtained under each of the models above.

4.5 Comparing Models Empirically

This comparison addresses two questions. First, how does the number of training instances affect the models' generalization to test data? Second, how does the *sampling strategy* employed—the choice of solver runs made in training—affect their relative accuracy?

To answer the first question, the models are compared as increasingly many instances in the training set are used for training. To answer the second question, the same experiment is conducted under multiple sampling strategies. Two distinctions between sampling strategies are particularly important. First, a strategy can run solvers the same number of times on every instance, or it can run them more often on some instances than on others. Second, if a strategy calls for different numbers of runs, it can run all solvers systematically the same number of times on an instance, or it can run different solvers separately different numbers of times on an instance. Three strategies are therefore employed: making two runs systematically on all training instances ("[2] (Sys.)"); making four runs systematically on half of all training instances and no runs on the rest ("[4, 0] (Sys.)"); and doing so separately

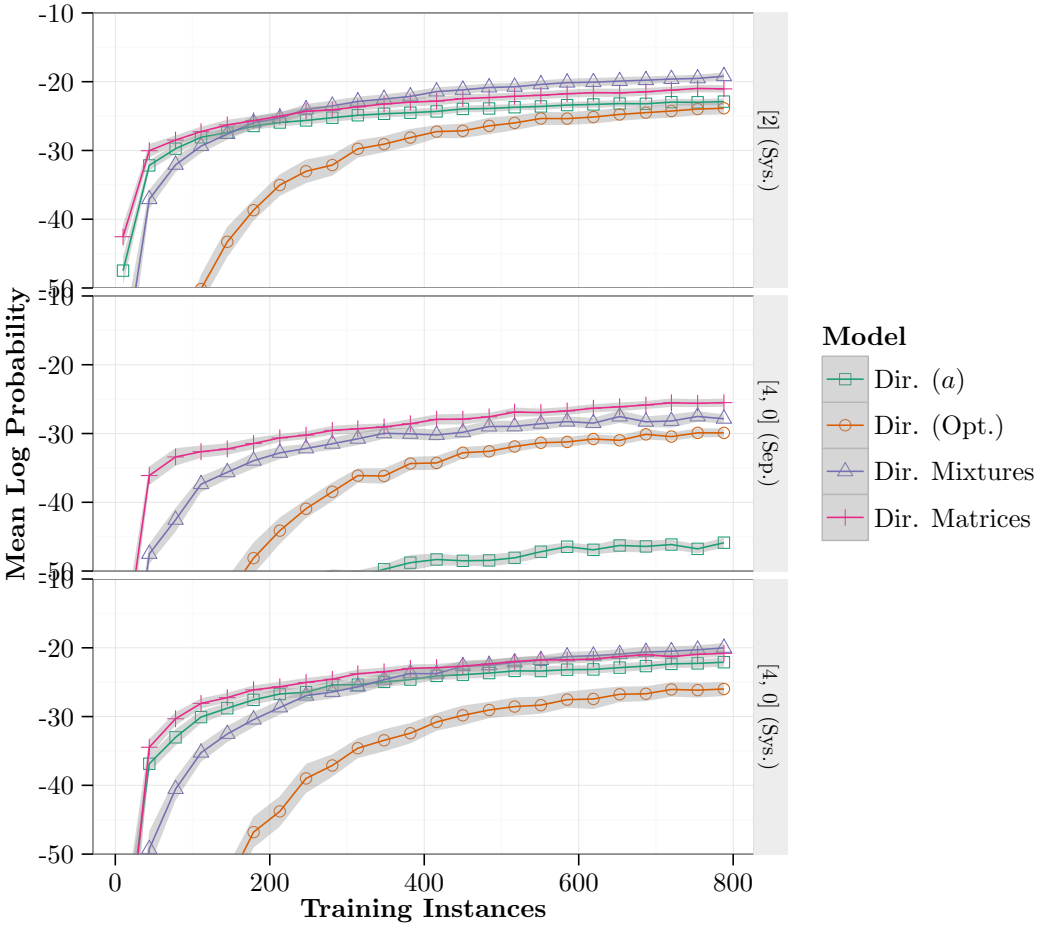


Figure 4.5: The mean log probability of test data under each of the models described in Section 4.3, averaging over 32 random 90% train/test splits, varying the number of training instances, and applying three different sampling strategies. Better models assign higher probability to the test data. The independent mixture prior ($K = 4$) works best under systematic sampling (top and bottom). The linked mixture prior ($K = 32$) allows the model to acquire information about an RTD of one solver from the runs of other solvers, and is the best model of data acquired by non-systematic sampling (middle).

for each solver ("[4, 0] (Sep.)").

Figure 4.5 presents the results of these experiments. Each sampling strategy highlights a relative strength of each model. When solvers are run systematically, the independent mixture prior has sufficient information about every instance to reconstruct RTDs, and it predicts the test data well. When solvers are run non-systematically, the linked-prior model performs best. The independent prior is less able to infer RTDs from which it has no samples, but the linked prior can use runs made by other solvers on the same instance to gain relevant information. Each model may therefore be useful, depending on the task to which it is applied and the structure of available training data.

4.6 Visualizing Model Output

Visualizing each model's estimates, even on a single instance, can provide additional intuition about their differences. Figure 4.6 presents such an example, on which the latent structure identified by the linked mixture model allows it to infer the RTD of a solver from which no runs were observed.

It is also informative to examine the high-level structure that such models infer in the data. To this end, Figs. 4.7 to 4.8 visualize the latent Dirichlet parameters fit by EM under the same model. These class parameters represent only *prior* information shared among instances. They do not control performance estimates directly: if many runs are made on an instance, those observations may outweigh the classes' influence on the estimated RTD. Nonetheless, the performance patterns they encapsulate match knowledge of the solvers and instances involved. For example:

- Component 2 describes challenging instances on which no solver succeeds.
- Component 9 describes instances easy for incomplete SLS solvers such as `gnovelty+`, but difficult for complete DPLL-derived solvers such as `minisat20`. The solver `kcnfs04`, DPLL-derived but explicitly designed for the random k-SAT instances

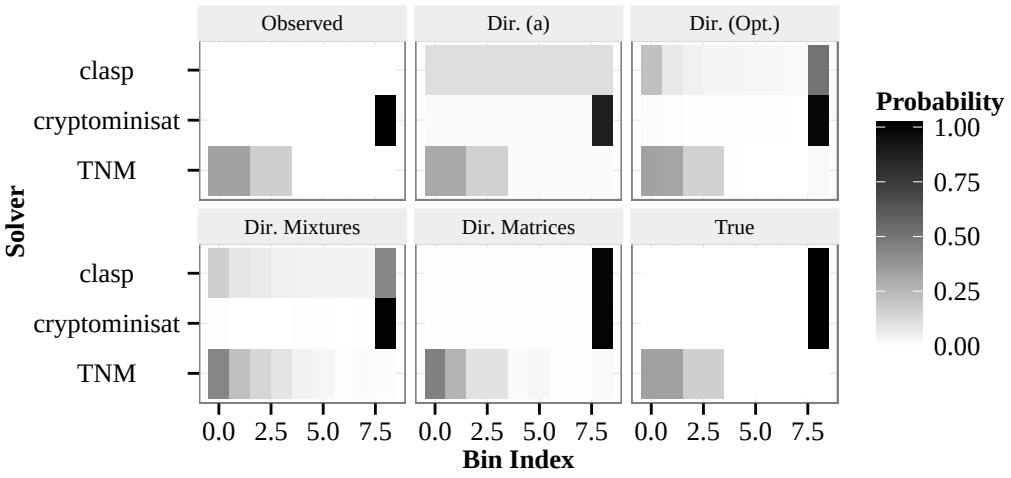


Figure 4.6: The MAP estimate under each model of `ppfolio` subsolvers' RTDs on SAT 2009 competition instance `unif-k3-r4.2-v14000-c58800-S1339275181-076.cnf`. Failed runs fall in bin 8. Four runs from `TNM` and `cryptominisat` and no runs from `clasp` were observed. Important differences between the models are evident. The uniformly smoothed model ("Dir. (a)") cannot provide any information about the RTD of `clasp` on this instance. If optimized asymmetric Dirichlet priors are used ("Dir. (Opt)"), the RTD becomes `clasp`'s average performance; note, also, that this stronger prior incorrectly places some mass on the failure bin of `TNM`. The independent mixture prior ("Dir. Mixtures") provides a similar estimate of `clasp`'s performance, but does not overly smooth the RTD of `TNM`. Since `clasp` rarely succeeds on instances easy for `TNM`, the linked mixture prior ("Dir. Matrices") can use that pattern to better estimate its RTD. The best choice of model thus depends on the sampling strategy employed.

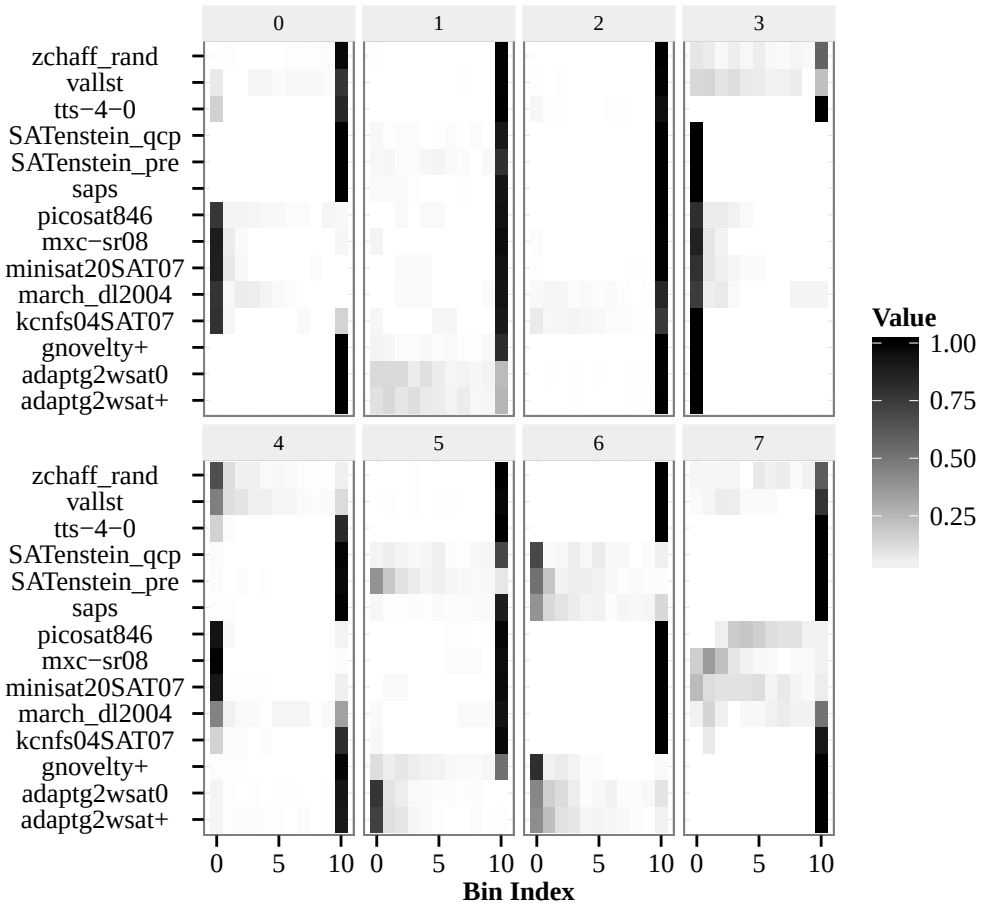


Figure 4.7: Latent classes inferred by the linked-component mixture model of Section 4.3.5, using $K = 16$ mixture components and $B = 10$ performance bins. Solver names have been shortened when necessary. Continued in Fig. 4.8.

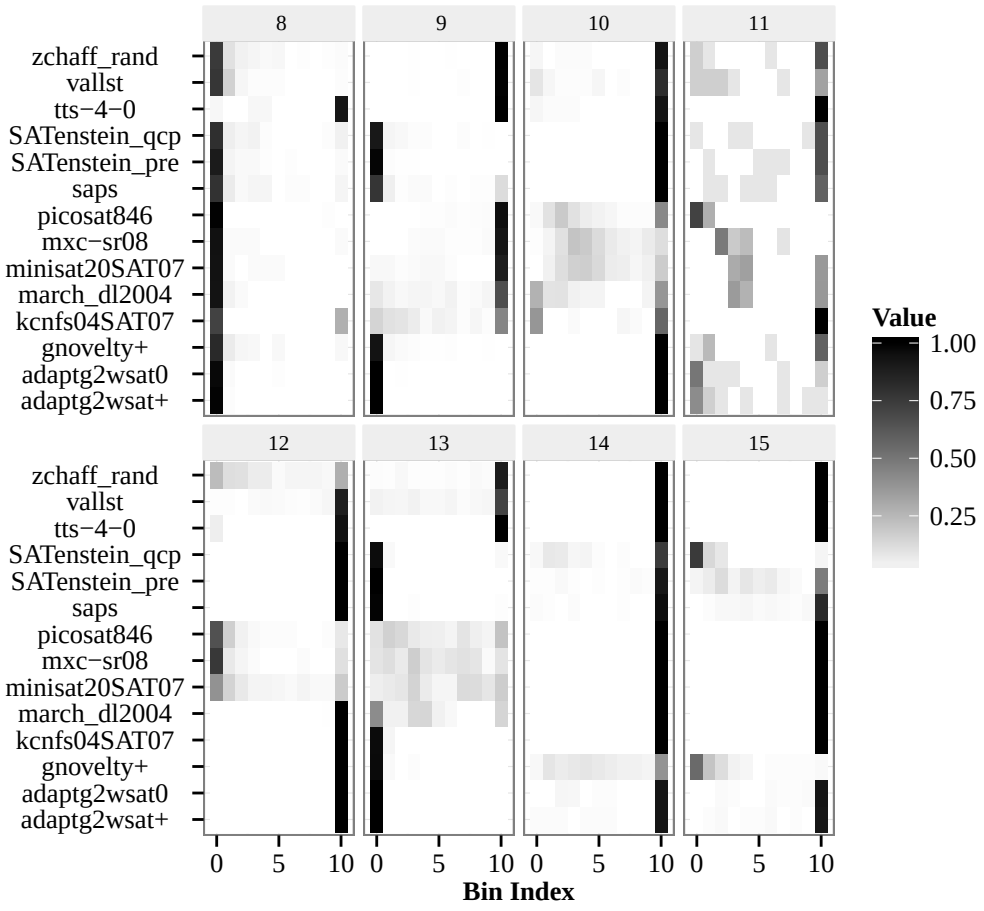


Figure 4.8: Latent classes inferred by the linked-component mixture model of Section 4.3.5. The structure it identifies in the solver performance data can be observed; these observations are expounded in the text.

on which SLS solvers excel, shares some of the performance characteristics of both families (Dequen and Dubois, 2004).

- Components 0 and 4 complement component 3: they describe instances easy for most complete solvers, but difficult for SLS solvers.

- Component 3, in particular, shows the older solvers `zchaff` and `vallst` struggling to match the performance of other solvers, even on instances easy for both complete and incomplete solvers.

The model's performance expectations are justified by domain knowledge. The model, however, discovers these expectations automatically, and quantifies them on particular instance distributions.

Another way to examine this model is through the responsibility its components take for instances across the data set. Those responsibilities can be compared to other domain knowledge about those instances, specifically, knowledge of the category to which each instance belongs. Figure 4.9 plots the share of responsibility each mixture component takes for each instance, i.e., the value of each β_k conditioned on each set of observations. With the instances reordered to group those of the same class, Fig. 4.10 shows that the correlation between an instance's class and its category becomes apparent. The structure identified by the model has, therefore, further justification in knowledge of the domain.

4.7 Modeling for Exploratory Visualization

Models of solver performance can also be applied for purposes outside the immediate goals of MAPP. One purpose which this dissertation explores is the human interpretation of performance data. The volume of data generated by a typical solver competition, the outcomes of dozens of solvers running on hundreds of instances, makes it difficult to uncover interesting aspects of performance buried in tabular results. How are instances organized in solver performance space? Are there instances on which all solvers perform poorly? Are

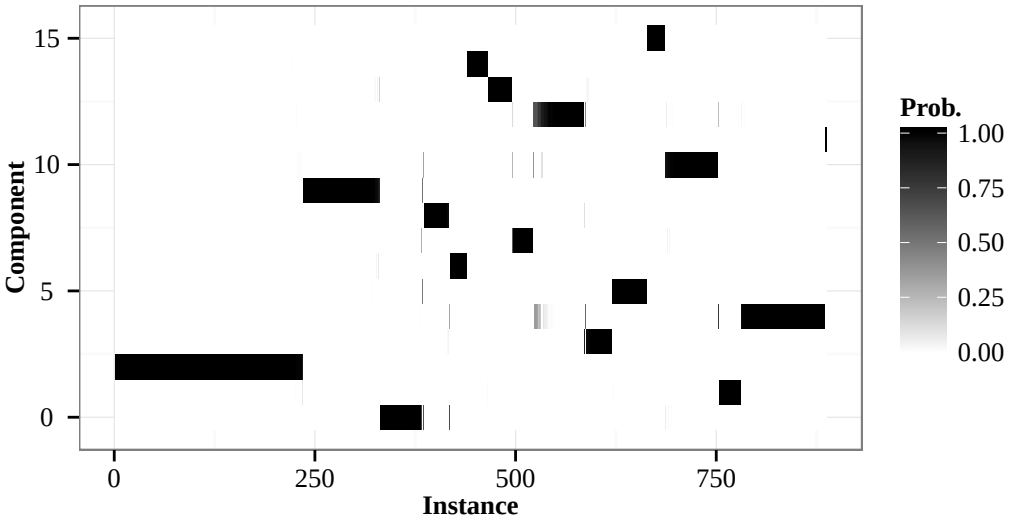


Figure 4.9: The responsibilities assumed for each SAT 2007 competition instance by each component of the model also visualized in Fig. 4.7. The same set of values underlie both figures. Each row corresponds to a mixture component, and each column corresponds to an instance. To highlight patterns in the data, the matrix has been reordered to minimize inter-column Euclidean distance (Hahsler et al., 2008). This reordering has the effect of grouping instances that belong to the same components. Most instances, but not all, are the responsibility of only one component. The model has thus rigidly divided the space of solver performance, sharing statistical strength only amongst instances it deems similar. Figure 4.10 shows that these classes correlate with known facts about instance performance.

there important differences between solvers' performance on satisfiable and unsatisfiable instances? Instance clustering and performance estimation, both of which are products of applying a mixture model to these data, can make it easier for humans to understand competition results and answer such questions.

The `borg-explorer` tool, part of the `borg` system in which MAPP has been im-

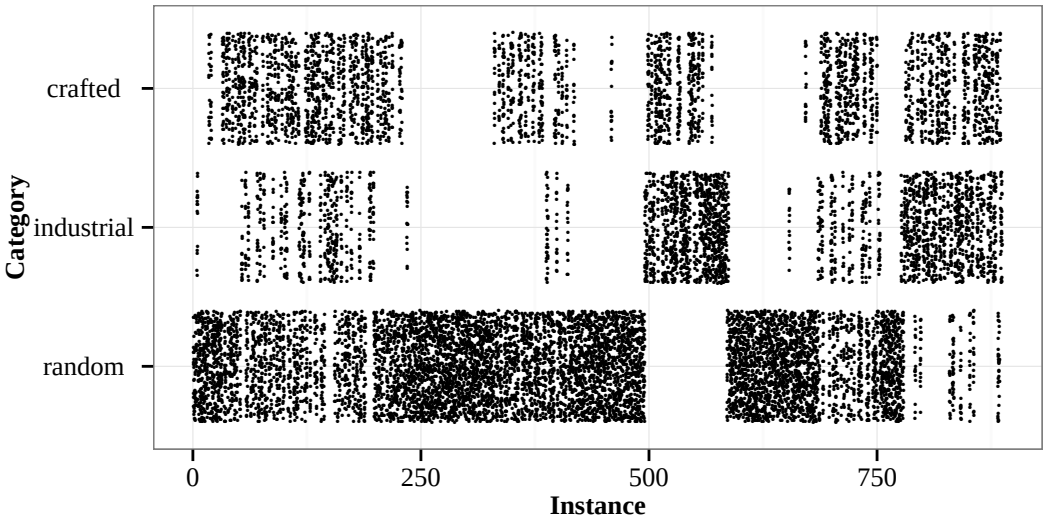


Figure 4.10: The competition categories of each instance, presented in the same order as in Fig. 4.9, i.e., so that instances belonging to the same component are adjacent. Each point represents an instance, and they are grouped into horizontal bands by category. Within a band, points are spaced vertically to make them more visible. Instances in a band tend to cluster together, meaning that the competition category and primary model component of an instance are correlated. This effect is particularly evident for the RANDOM and INDUSTRIAL categories. Instances in the CRAFTED category, which contains an assortment of challenging instances of various kinds, cluster less easily according to solver performance. This visualization shows that the mixture model is naturally recovering some of how a human might organize the data.

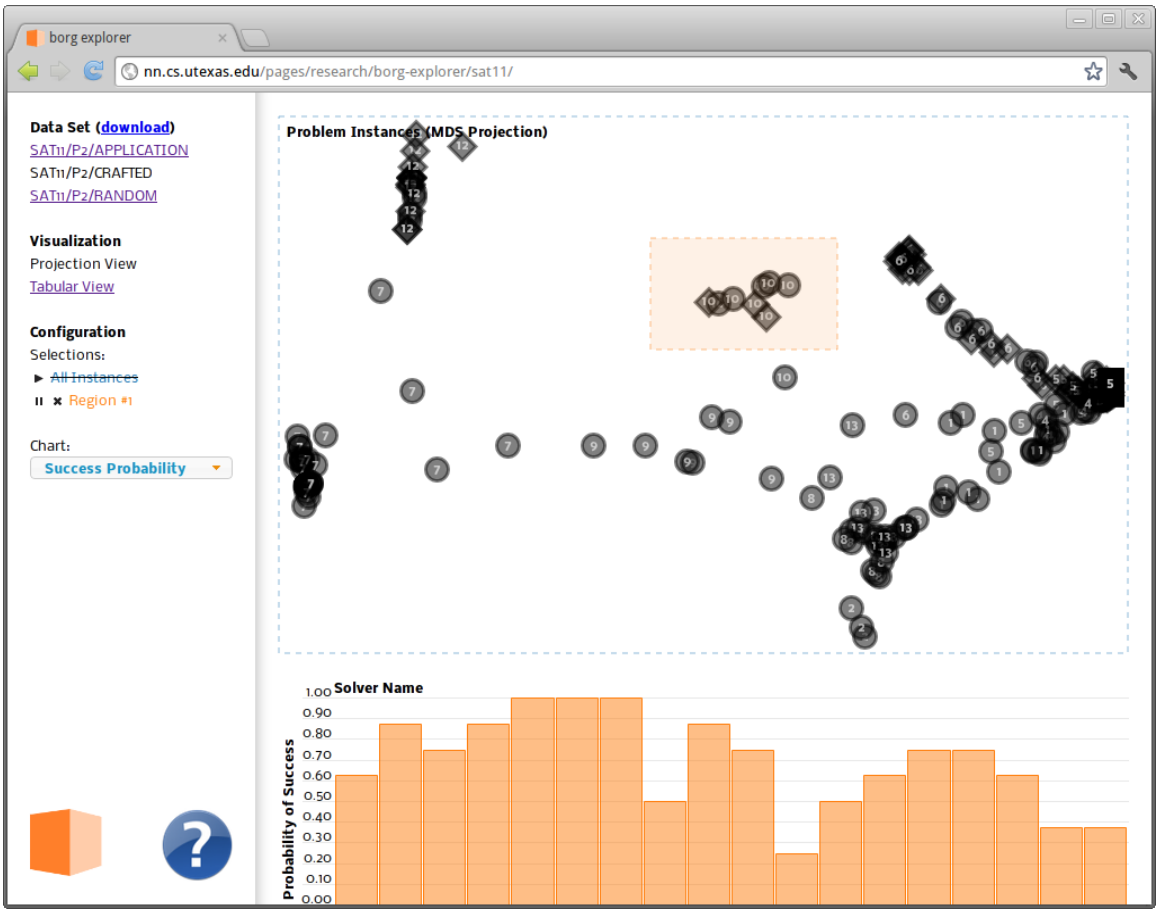


Figure 4.11: Screenshot of the `borg-explorer` visualization tool in use. This web-based tool leverages a mixture model of performance, such as that examined in Figs. 4.7 to 4.10, to provide a picture of solver performance more accessible than the raw data and more detailed than a simple solver ranking. Each circle represents a satisfiable instance, and each diamond an unsatisfiable instance; each symbol is numbered with the latent class to which it predominantly belongs; and symbols are positioned to maintain, as much as possible, distances between them proportional to the differences in the estimated RTDs. The performance of solvers in selected regions of this space can be plotted. The tool makes high-level patterns evident: the performance differences between clusters 7 and 12 in the left half of the space, for example, can be seen as also correlated with a satisfiable / unsatisfiable split.

plemented, was designed to make such exploration possible. As seen in Fig. 4.7, it provides an interactive, high-level picture of solver behavior on large benchmark collections by laying out instances in a two-dimensional space according to performance. Visualization environments generated by the tool are available online[1][2], and can used to explore the results of the 2009 and 2011 competitions. After its introduction, interest in `borg-explorer` drove its integration by the developers of EDACC, an environment for conducting rigorous solver experiments on cluster hardware (Balint et al., 2010).

While designed to visualize competition data, `borg-explorer` could be used for any solver performance data set. Virtually every paper on solver development, for example, includes an empirical analysis of a new technique's performance. Tools such as `borg-explorer` might allow researchers to interactively explore such results. The ability of models to aid human understanding thus provides another reason for their continued development in this domain.

4.8 Conclusions

The core of MAPP is a set of solver performance distributions estimated for training instances. Since these estimates must be made from a limited number of observed solver runs, this chapter introduced a family of discrete mixture models of solver performance. Estimating values for the latent variables in these models, using EM, corresponds to estimating the run time distributions underlying solver performance. On data collected from a standard suite of solvers running on benchmark instances, various strengths of models are apparent: simple models with minimal structure are robust and effective when data are plentiful, while models with more complex priors can be useful when data are scarce or missing. The modeling component of MAPP thus provides an essential link between the portfolio's raw training data and the beliefs on which it bases its actions. The following

[1] `http://nn.cs.utexas.edu/pages/research/borg-explorer/sat09/`

[2] `http://nn.cs.utexas.edu/pages/research/borg-explorer/sat11/`

chapter develops and evaluates a method for combining instance feature information with those beliefs, adjusting them to better suit each test instance.

Chapter 5

Integrating Instance Features

There are three sources of information available to a portfolio: past solver runs on training instances, past solver runs on the current instance, and the superficial appearance of that instance. Chapter 4 and Chapter 6 allow MAPP to make use of the first two sources of information. This chapter handles the third. Its feature integration approach allows MAPP to tailor its decisions to particular instances before running a solver. It leverages existing supervised classifiers, and provides multiple performance scenarios to planning. This chapter lays out the details of feature integration in MAPP, evaluates its effectiveness, and analyzes its operation.

5.1 Motivation

MAPP uses the models developed in Chapter 4 to acquire knowledge of solvers' performance distributions on training instances. Even perfect knowledge of these distributions, however, leaves the portfolio method knowing only how solvers performed in the past. When faced with a new instance, MAPP must somehow apply this knowledge of the past toward making a judgment about the future. In particular, the portfolio must predict how solvers are likely to perform on that instance given only information about its superficial

appearance. Does it include many clauses? Are those clauses tightly linked? Such details can, when coupled with experience, offer clues to how well different solvers will perform.

Algorithm-selection portfolios often leverage these features by training supervised learners to predict solver performance directly—but, because the notion of performance they employ is limited to an aggregate score, such portfolios cannot construct or benefit from solver execution schedules. In contrast, so-called "non-model-based" portfolios do build such execution schedules, tailoring them to an instance's neighborhood in feature space—but the relevance of that neighborhood depends on the fortuitous suitability of the feature set.

MAPP leverages instance features in a way that combines the strengths of both approaches. It does so by using features to predict the similarity between instances, and, critically, by defining that similarity in terms of solver performance. As in algorithm-selection portfolios, this approach builds on the mature field of supervised learning, and it trains discriminative models specifically to associate feature values with properties of solver performance, and naturally to ignore irrelevant features. As in non-model-based portfolios, the outcome of this integration is the ability to connect training instances to previously-unobserved test instances. That connection makes it possible to create solver execution schedules. The fundamental principle of MAPP, the explicit representation of uncertainty as a weighted collection of performance scenarios, is also maintained.

5.2 A Review of Instance Features

Instance features are arbitrary, efficiently-computable properties of the members of a given problem domain. Features are an important aspect of most successful portfolio methods (Xu et al., 2008a; Kadioglu et al., 2011; Gebser et al., 2011b). Commonly-used features in SAT include the number of variables, counts of particular clause structures, summaries of local search trajectories, and statistics of the constraint graph (Nudelman et al., 2004). Such features are often supplied to a regression method in order to predict a performance score, such

as expected run time, for every solver. This approach is employed by the `claspfolio` portfolio for ASP, which uses a support vector machine (SVM), and by `SATzilla2009`, which uses ridge regression. In SAT, MAPP uses a subset of the standard feature set introduced by Nudelman et al. (2004), as presented in more detail in Chapter 7.

Features are useful because they provide information about solver performance. This connection, however, may be indirect and situation-dependent. Even the file path of an instance, for example, could provide information about its provenance, the encoding scheme that generated it, and thus its difficulty. It will do so, however, only within a single environment. Commonly employed features are less arbitrary, but their meaning may still be sensitive to the distribution of instances at hand.

Before diving into the process by which such features are used in MAPP, it may therefore be instructive to focus briefly on an example feature and to describe the information it conveys. The prototypical feature, as mentioned in Section 3.2.2, is the ratio of clauses to variables in CNF SAT. As clauses are added to an instance of random k-SAT, the formula becomes increasingly constrained and fewer satisfying assignments exist. At a high ratio of clauses to variables, then, the formula is likely unsatisfiable. At a low ratio, it is likely satisfiable. Between these extremes, there is a point where random formulae have either few or zero satisfying assignments, and the addition or subtraction of a clause tips an instance to one side. This point has been deemed the *phase transition* in random SAT. In 3-SAT, the phase transition lies near a ratio of 4.26 (Crawford and Auton, 1996).

For a distribution of random k-SAT instances, this ratio is not of exclusively scientific interest: it also provides information that can help to solve those instances. If an instance appears largely unconstrained, a short run of an SLS solver is likely sufficient to identify a satisfying assignment. If an instance appears highly constrained, a short run of a DPLL-derived solver is likely sufficient to prove unsatisfiability. Instances near the phase transition may require much longer runs of both solver types. Knowledge of this instance feature thus enables solution strategies to be specialized for particular instances, by provid-

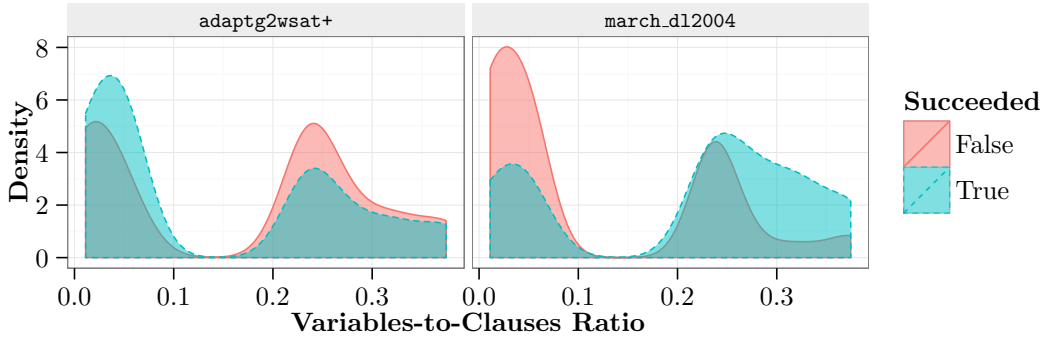


Figure 5.1: The proportion of successful and unsuccessful runs of two different solvers on SAT 2007 RANDOM instances, as a function of those instances' variables-to-clauses ratio, estimated by KDE. In some cases, this ratio expresses how constrained a formula is; and complete solvers like `march_dl2004` often perform better than incomplete solvers like `adaptg2wsat+` on highly constrained instances, which may be more likely to be unsatisfiable. In this case, however, the ratio expresses a different aspect of the instance distribution, and our expectations about its import are incorrect. This example illustrates the difficult, empirical nature of connecting feature values to predictions about solver performance.

ing information about solver performance on a new instance *before* the first solver run has been made.

The clauses-to-variables ratio, however, is only one of many possible features. It is most useful on collections of random k-SAT instances when k is fixed. On structured instances, other features may be more valuable. Detailed statistics of the constraint graph, for example, may suggest satisfiability or unsatisfiability in situations where the clauses-to-variables ratio does not. Furthermore, even that well-studied feature can exhibit unexpected patterns. Figure 5.1 plots the run times of a complete and an incomplete solver against the reciprocal of this ratio on the RANDOM instances used in the 2007 competition. In

this situation, unexpectedly, the incomplete solver succeeds more often at higher values of the clauses-to-variables ratio, and the complete solver succeeds more often at lower values. Expectations fail to match observed behavior because these particular random k-SAT instances were generated over a range of k, rather than over a range of clause counts.

The precise connection between features and solver performance can, therefore, quickly become an empirical question, and one that depends on the composition of a particular instance mixture. The next section thus applies the machinery of supervised learning to the specific set of performance and feature data at hand.

5.3 Feature Integration as Classification

MAPP leverages features by associating them with distances between instances. In this approach, multiple discriminative models predict the distance, according to some metric, between a new instance and every training instance. Unlike the nearest-neighbor approach of portfolio methods like those of Malitsky et al. (2011) and Nikolić et al. (2011), which measure distances in feature space, MAPP measures distances between instances in solver performance space. This difference makes feature integration in MAPP more robust to differences between the two spaces. This section lays out the integration scheme in detail.

Recall the central modeling assumption of Chapter 4, that solvers' RTDs on a test instance are identical to their RTDs on some training instance. This assumption might suggest one possible goal for feature integration: to identify groups of instances on which solvers perform identically, based on the features of those instances. In reality, however, the assumption of performance-identical instances does not hold perfectly; solvers' behaviors on different instances are rarely truly the same. Instead, for the purpose of feature integration, the goal of finding identical instances can be relaxed into the closely-related goal of finding *most-similar* instances.

For a test instance with feature vector ϕ, then, the goal of feature integration is a weighting over training instances that expresses their likely similarity to the test instance,

measured according to the similarity of their RTDs. An RTD-based distance between two instances with sets of RTDs $\{\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(S)}\}$ and $\{\boldsymbol{\beta}^{(1)}, \ldots, \boldsymbol{\beta}^{(S)}\}$ can be defined as

$$f(\boldsymbol{\theta}, \boldsymbol{\beta}) = \frac{1}{S} \sum_{s=1}^{S} \left[\max_{b \in \{1,\ldots,B\}} \left| \sum_{i=1}^{b} \boldsymbol{\theta}_i^{(s)} - \sum_{i=1}^{b} \boldsymbol{\beta}_i^{(s)} \right| \right], \tag{5.1}$$

i.e., analogously to the standard Kolmogorov-Smirnov test, as the mean of the maximum differences between their corresponding per-solver RTDs represented as cumulative distribution functions (cdfs).

In practice, there is little value in predicting distances between dissimilar instances precisely; a portfolio method gains more value from focusing on similar instances than it does from estimating long distances exactly. Features, then, are used in MAPP to predict the probability that each training instance is one of the M training instances whose RTDs are most similar to those of the test instance. This goal can be cast as a classification problem: for each training instance, M training instances are labeled as similar to that instance, and the remaining as dissimilar. A binary classifier is then trained to predict those class labels. This process builds one classifier for each instance in the training set.

For a given test instance, the probability of class membership predicted by each classifier becomes the weight of consideration given to its associated training instance during planning, as developed in Chapter 6. The supervised learning method applied in this dissertation is l_2-regularized logistic regression, the standard first choice in binary classification (Hastie et al., 2009). It is straightforward, efficient, and widely-used. It assumes that features are independent and contribute additively to a binary class probability through the nonlinear logistic function. It is this class probability that is taken as each instance's weight.

Early experiments found one small modification to typical logistic regression to be important. Since few instances receive the most-similar label relative to the number of training instances as a whole, the distribution of labels is skewed. To force logistic regression out of the baseline strategy of labeling every instance as dissimilar, the two classes are treated differently: in training, false negatives are penalized ten times more highly than

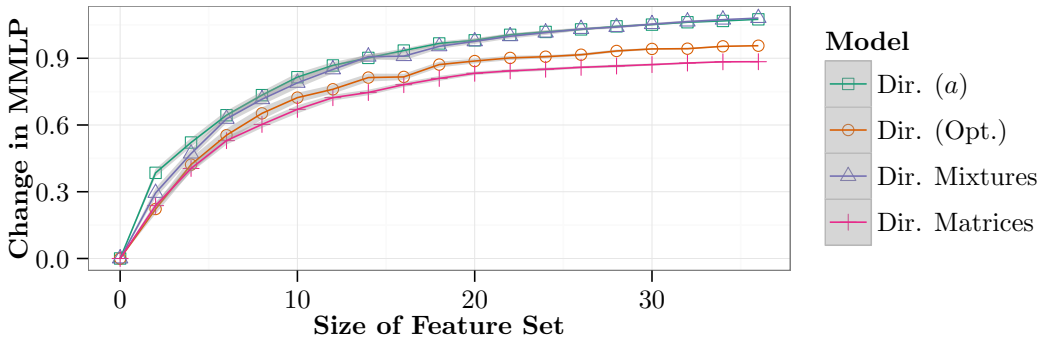


Figure 5.2: The change in the mean log probability of held-out test data after including instance feature information. Single models were trained on a random 80% subset of the 2007 SAT competition instances, then predictions made and results averaged over randomly-selected feature sets of varying size. The static features of Nudelman et al. (2004) were used, excluding expensive clustering coefficient information. Classifiers were trained on $M = 32$ positive examples. Features measurably improved predictions. The results in Chapter 7 will further show that this improvement is capitalized in the problem-solving performance of the portfolio.

false positives. This class-specific weighting is a standard feature of most implementations of logistic regression (Pedregosa et al., 2011; Fan et al., 2008).

The ability to leverage the standard tools of supervised learning in this way is an important advantage of MAPP's approach to feature integration. If important feature interactions are known, for example, a more expressive classifier could be applied without changing any aspect of the surrounding framework. More generally, supervised learning is a well-studied area; it is reassuring to know that its methods can be brought to bear.

	Change in MMLP		
Year	**INDUSTRIAL**	**CRAFTED**	**RANDOM**
2007	0.53	0.54	0.90
2009	0.41	0.66	1.05

Table 5.1: The reduction in uncertainty achieved by including feature information on each 2007 and 2009 competition category, in terms of the change in the mean log probability of held-out runs of the `SATzilla2009` suite. Results are averaged over random 80% train/test splits. Features benefit the RANDOM category the most and the INDUSTRIAL category the least, an effect consistent with knowledge of the SAT/UNSAT split that is most important for solvers on RANDOM. Features are, however, measurably useful on every category.

5.4 Experiments

While evaluations in Chapter 7 will consider the effect of feature integration on the performance of MAPP as a whole, experiments in this chapter examine it entirely through its effect on the probabilities assigned to test data. These experiments therefore focus directly on the reduction in uncertainty that features bring, with uncertainty expressed by the mixture-of-instances representation that forms the core of MAPP. In this representation, uncertainty is reduced by producing a weighting over the training instances that favors more strongly those with more performance similarity to a new instance.

Figure 5.2 shows results of an experiment compares mean scores under feature subsets of varying size. Although certain models appear to respond more strongly than others, log probability shows clear improvement in every case as the classifiers gain access to more feature information. That information, in other words, measurably reduces uncertainty. Table 5.1 extends this measurement to specific categories: features remain consistently useful,

albeit not equally so. Extensive experiments in Chapters 7 to 8 will confirm that this reduction in uncertainty makes MAPP substantially more successful. Furthermore, the curve of improvement suggests that, while some features may be more valuable than others, most features contribute some piece of useful information.

A related observation can be drawn from Fig. 5.3, which visualizes the weight each classifier gives to each feature. Different features appear useful for identifying different areas of similar solver performance. This observation further motivates going beyond pure feature-space similarity, as in the k-NN approach, by connecting features to the true goal of predicting performance. It also suggests that feature-selection approaches, like that of Kroer and Malitsky (2011), cannot completely mitigate drawbacks of k-NN-based performance prediction by focusing on fewer features; such an approach still employs the same feature set across the entire space.

The supervised scheme, as described, should be more robust to misleading features. To test this hypothesis, Fig. 5.4 plots the change in the feature-conditioned log probability of test data as garbage features are added to the feature set. Indeed, k-NN is less predictive as the feature space is corrupted, while the similarity-based scheme declines only minimally. This scheme thus makes it simpler to construct a MAPP portfolio in a new domain, since features may be devised, implemented, and applied without separately evaluating their utility.

5.5 Conclusions

Leveraging information from superficial instance features allows a portfolio to acquire more knowledge of solvers' performance on an instance before it makes its first decision. It is, therefore, a valuable part of a portfolio method. Existing feature integration schemes operate by predicting independent solver performance scores, or by grouping neighboring instances in feature space. The former provides too little information for planning, and the latter is sensitive to arbitrary aspects of the feature set. The experiments in this chapter

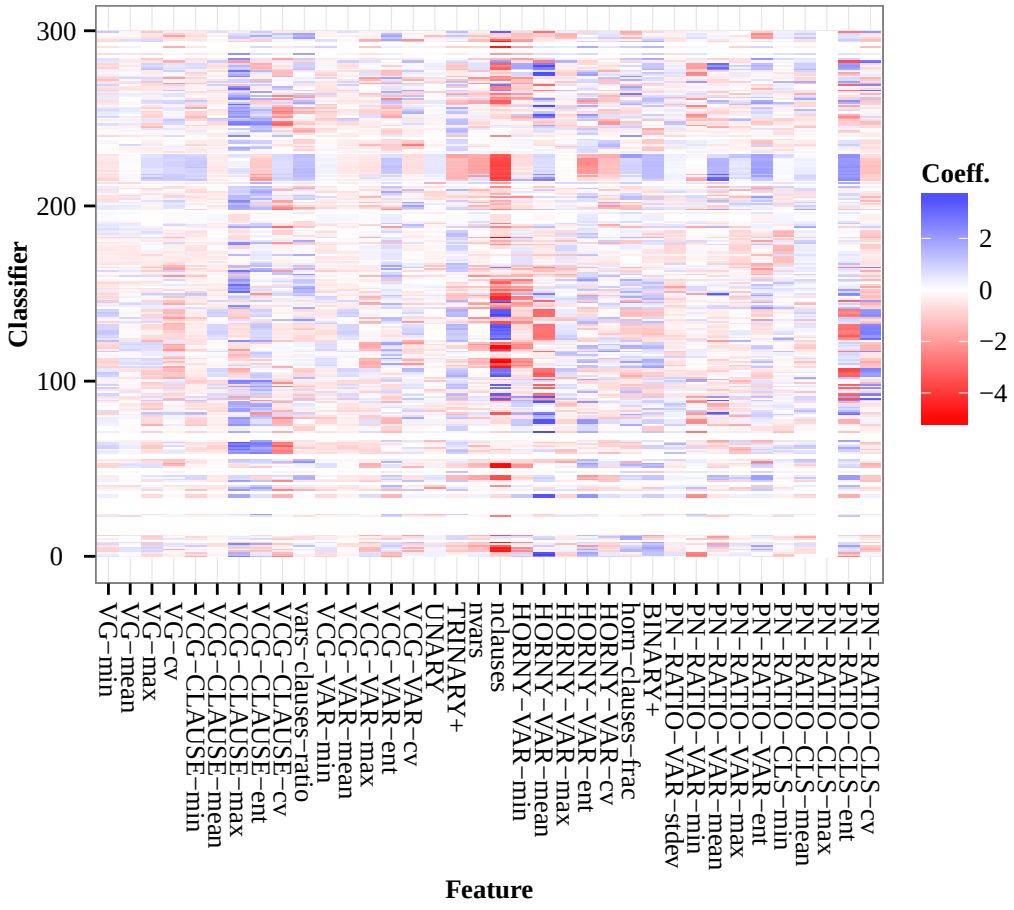


Figure 5.3: Values of the coefficients associated with each feature across per-instance classifiers for all 2007 instances. The order of classifiers is arbitrary. Feature values were scaled to zero mean and unit variance. Some features, such as the raw clause count `nclauses`, can be seen receiving higher average weight than others. Classifiers associated with different instances, however, use different sets of features; some, for example, give `nclauses` zero weight. This observation suggests that different sets of features are relevant to distinguishing different regions of this solver performance space.

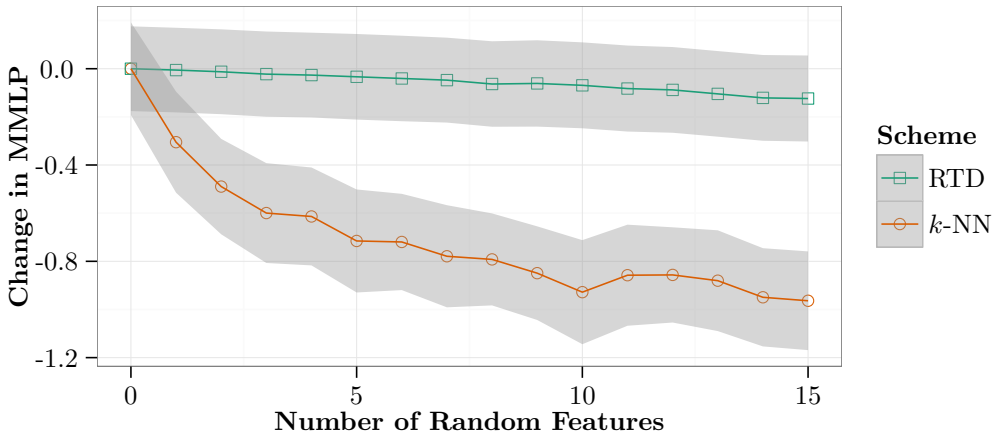


Figure 5.4: The change in the mean log probability of held-out run data according to mixtures of training instances weighted by k-NN and by RTD-based feature integration, as random features are added to the feature set. Values were averaged over 64 random 80% train/test splits of 2007 competition instances, and runs were of the `SATzilla2009` suite. For both schemes, $M = 32$ nearest neighbors were used. An approach that weights instances solely according to their proximity in feature space, as k-NN does, is sensitive to the feature set employed. In contrast, discriminative models of RTD similarity are trained to ignore non-predictive features, and are thus less likely to be misled.

showed that the alternative scheme employed in MAPP combines the strengths of both approaches.

Chapter 4 established the initial state of the MAPP belief representation, and this chapter enabled feature information to influence it. The fundamental principle of MAPP—the representation of portfolio information as a weighted mixture of possible performance scenarios—thus remains an effective, flexible way to manipulate and reason about performance uncertainty in the portfolio context. The next chapter shows that this representation can also be naturally employed in the computation of solver execution schedules.

The MAPP feature integration scheme, however, yields a weighting over all possible performance scenarios, i.e., over all training instances, of which there may be thousands. To fully leverage the information provided by instance features, then, a solver execution schedule should take all of those scenarios into account, not only a small neighborhood of them. This goal may be particularly important when features are unavailable, as in a new domain, or less informative, as they are on the INDUSTRIAL collections in the 2007–2011 SAT competitions. The efficiency of planning over many possibilities is therefore an important concern, and will be addressed in Chapter 6 as well.

Chapter 6

Planning Solver Execution

The preceding chapters constructed the components of MAPP that acquire information. These components extract performance distributions from limited run data, and leverage instance appearance to predict the similarity between such distributions. This chapter builds the third component of MAPP, which *uses* that information to solve computationally difficult instances given limited processor time. Its input is a suite of solvers and their estimated RTDs on a set of instances, and its output is a solver execution schedule that allocates that processor time to solver runs. The ideal schedule maximizes the expected number of instances solved; equivalently, given those solvers and a set of their possible RTDs on a single instance, it maximizes the probability of solving that instance.

As mentioned in Chapter 3, Streeter et al. (2007a,b) reduced this run scheduling problem to the min-sum set cover problem (Feige et al., 2004). They further developed a greedy approximate portfolio scheduling algorithm, and showed that any better approximation would be NP-hard. This chapter will nonetheless improve on their result by reducing instead to the knapsack problem and leveraging dynamic programming to arrive at a *pseudo*polynomial-time solution that remains practical in the context of MAPP.

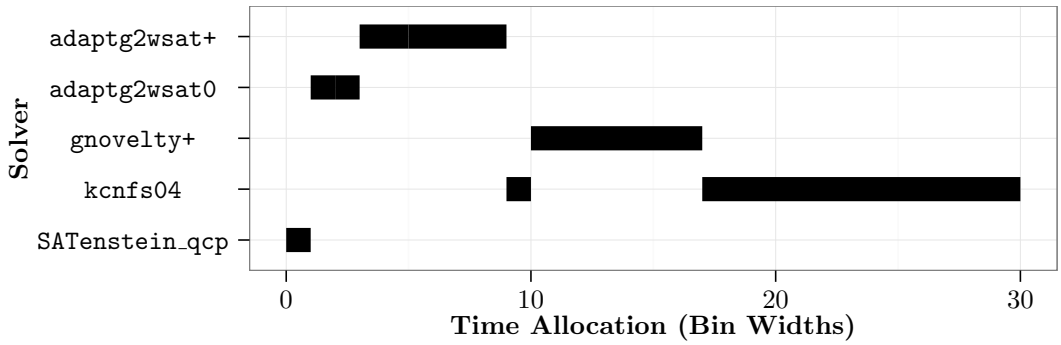


Figure 6.1: An example run schedule, including restarts, computed for the set of `SATzilla2009` subsolvers on the set of all instances from the RANDOM category of 2007 SAT competition. The goal in computing such a schedule is to maximize the number of instances it will solve in that set. Common tradeoffs in solver scheduling are visible: the plan mixes short runs to quickly solve easy, satisfiable instances with longer runs to solve more difficult, unsatisfiable instances, and it spends the largest amount of time executing a general-purpose CDCL solver effective at both tasks.

6.1 Planning for a Known Run Time Distribution

A run schedule can be represented as a sequence of T solver-duration pairs $\langle (s_1, b_1), \ldots, (s_T, b_T) \rangle$ where each s_t is the index of a solver and each b_t is the index of a particular run duration, typically associated with a model bin. An example of such a schedule is presented in Fig. 6.1. A planning algorithm constructs a schedule subject to a constraint on the total runtime $\sum_{t=1}^{T} d_{b_t} \leq C$, for some budget or "cutoff" C, where d_t is the upper end of the run duration range spanned by bin t.

First, consider scheduling solvers on a single instance for which their run time distributions are known. The models in Chapter 4 estimate the probability $p(b_{sir} = j)$ that run r of solver s on instance i would fall into bin j of B discrete bins, with a designated bin

receiving only all failing runs. Each run is independent: conditioned on the choice of solver, instance, and duration, a run's outcome depends only on the solver's internal randomness. The overall probability of a plan's success does not, therefore, depend on the order of its actions. Furthermore, if the RTDs are known, the outcome of a run provides no additional information about the unobserved outcomes of later runs.

Planning in this setting can be cast as a variant of the classic unbounded knapsack problem (UKP), and this interpretation leads to an efficient algorithm for computing a run schedule. The UKP asks for the selection, made with replacement from a set of items that are each associated with a "value" and a "weight", that maximizes the total value subject to a maximum total weight (Kellerer et al., 2004). Formally, this goal is to

$$\text{maximize} \sum_{a=1}^{A} v_a x_a, \tag{6.1}$$

$$\text{subject to} \sum_{a=1}^{A} w_a x_a \leq d, \tag{6.2}$$

where A is the number of items, v_a the value of item a, w_a the weight of item a, and x_a the number of copies of item a that were selected. In solver scheduling, a selection is made from a set of possible runs, each associated with a solver and a duration, subject to a maximum total duration. That selection should minimize the probability that every selected run takes longer than the duration allotted to it. If bin B is assumed to be the failure bin, and if bins are equally spaced, so that a bin index j can be seen as a duration expressed in bin widths and B taken as the cutoff, then this objective can be written as:

$$\text{minimize} \prod_{s=1}^{S} \prod_{j=1}^{B-1} p(b_{si\cdot} > j)^{x_{sb}}, \tag{6.3}$$

$$\text{subject to} \sum_{s=1}^{S} \sum_{j=1}^{B-1} j x_{sb} \leq B. \tag{6.4}$$

The correspondence between the knapsack problem and the solver scheduling problem can be made even more evident by rewriting the objective in Eq. (6.3) in terms of total log

probability, as

$$\text{minimize} \sum_{s=1}^{S} \sum_{b=1}^{B-1} x_{sb} \log p(b_{si\cdot} > j), \tag{6.5}$$

subject the same constraint, Eq. (6.4). Thus solver scheduling is an instance of the knapsack problem where the items consist of solver-duration pairs, the objective measures failure probability, and the goal is minimization instead of maximization.

Note that the probability $p(b_{si\cdot} > j)$ of a run of solver s failing on instance i, i.e., falling into a bin representing more time than bin j, is the inverse of the cdf known as the *survival function*. Survival analysis is a rich field whose connections to algorithm portfolios have been explored more deeply by Gagliolo and Schmidhuber (2006), also discussed in Section 3.4.

The standard knapsack problem, and thus—when the true probabilities are known—the solver scheduling problem, can be solved exactly via dynamic programming (Bellman, 1957). Call the minimum survival probability with m bin intervals remaining p^*_{mi}, and consider the situation when time remains for only a single, short run. Then the objective becomes:

$$\text{minimize} \sum_{s=1}^{S} \sum_{b=1}^{1} x_{sb} \log p(b_{si\cdot} > j) = \sum_{s=1}^{S} x_{s1} \log p(b_{si\cdot} > 1). \tag{6.6}$$

The optimal choice in this situation is the solver most likely to terminate successfully in the first bin, so

$$\log p^*_{0i} = 0, \tag{6.7}$$

$$\log p^*_{1i} = \min_{s \in \{1,\ldots,S\}} \log p(b_{si\cdot} > 1), \tag{6.8}$$

and the survival probability with m bin intervals remaining is given in general by:

$$\log p^*_{mi} = \min_{s \in \{1,\ldots,S\}, j \in \{1,\ldots,m\}} \log p(b_{si\cdot} > j) + \log p^*_{(m-j)i}. \tag{6.9}$$

By computing each p^*_{mi} starting from p^*_{1i}, the minimum probability of failure on instance i, p^*_{Bi}, can be computed in time $O(SB^2)$. The plan associated with this value is the desired

optimal plan. Dynamic programming solves the knapsack problem in pseudopolynomial time, i.e., in time polynomial in the *value* of the inputs, but the problem remains NP-complete: it is believed to require time exponential in the *size* of its inputs.

In practice, a user wishes not only to maximize the probability of success within the cutoff, but also to solve instances quickly. Every ordering of actions in a plan computed as above will share the same probability of success, but some orderings will solve instances more quickly on average. A plan should therefore be reordered after it is obtained. In this dissertation, actions are sorted in decreasing order of the quantity $p(b_{si\cdot} \leq j)/j$, the cost-weighted probability of each action succeeding. This quantity is known as "efficiency" in the knapsack literature and elsewhere (Kellerer et al., 2004).

6.2 Planning for an Uncertain Run Time Distribution

If an instance's RTDs are known with certainty for every solver, the approach of Section 6.1 results in an optimal plan for solving that instance, even when solvers are nondeterministic. In general, of course, these distributions are not known. One way to express this uncertainty is as a set of RTD matrices, each providing one *possible* distribution for every solver, and each associated with a probability of representing the instance accurately. This approach integrates well with the models in Chapter 4, which characterize an RTD distribution as a finite mixture, and from which multiple candidate settings of their latent parameters can be obtained. As outlined in Chapter 1, this explicit representation of uncertainty is the core principle of the MAPP approach.

When uncertainty is represented by such a mixture, the problem of computing a schedule for one instance with unknown RTDs is equivalent to that of computing a schedule for many instances with known RTDs. The greedy planner of Streeter et al. (2007b) works with the latter formulation, and their results therefore apply directly to this setting as well. Adapted slightly to account for the probability associated with every RTD matrix,

this greedy planner chooses an action according to

$$g_j = \underset{a \in \{1,\ldots,S\} \times \{1,\ldots,T\}}{\arg\max} \frac{f(\langle g_1, \ldots, g_{j-1}, a \rangle) - f(\langle g_1, \ldots, g_{j-1} \rangle)}{j}, \tag{6.10}$$

where

$$f(\langle (s_1, b_1), \ldots, (s_T, b_T) \rangle) = \sum_{j=1}^{I} p(i = j) \left[1 - \prod_{t=1}^{T} p(b_{s_t j \cdot} > b_t) \right] \tag{6.11}$$

is the probability of a given plan succeeding. Intuitively, this greedy planner employs the same type of efficiency heuristic that was used to reorder actions in the knapsack plan: it starts from an empty sequence of actions, and repeatedly augments that sequence with the action that maximizes the probability of success per time step. Its greedy nature, however, means that this planner never revisits actions once they have been added to the plan.

To move beyond greedy planning, the dynamic programming approach can be extended to provide an approximation in the case of uncertainty. Equation (6.9) becomes:

$$\log q^*_{mi} = \min_{s \in \{1,\ldots,S\}, j \in \{1,\ldots,m\}} \log \sum_{l=1}^{I} p(i = l) p(b_{sl \cdot} > j) p^*_{(m-j)l}. \tag{6.12}$$

This algorithm chooses actions by considering their effects averaged over all performance scenarios. Each scenario is weighted by the predicted probability, computed by the distribution-similarity approach developed in Chapter 5, that it accurately represents the current situation. Since it proceeds by constructing multiple candidate plans, this approach is not tied to past decisions as rigidly as is the greedy planner.

However, since the algorithm computes a plan by working, in effect, backwards in time, it ignores the information-gathering effects of actions under uncertainty. The success or failure of the first action in a plan should alter the probabilities assigned to the scenarios considered, as observations affect belief state in any partially observable MDP—but the dynamic-programming paradigm means that those probabilities are taken as fact before the actions that affect them are selected. The algorithm is, therefore, suboptimal. It appears that suboptimality is the price paid for efficiency. Furthermore, unlike those of the

greedy planner, plans computed by the knapsack planner become optimal as knowledge of a solver's RTDs on an instance becomes certain: if only a single instance receives weight, the algorithm collapses to solving an instance of the standard UKP.

6.3 Experimental Evaluation

The knapsack planner offers compelling benefits in theory, such as optimal planning under known RTDs, but it is nonetheless approximate in general. Its performance relative to the greedy planner can be evaluated empirically. Figure 6.2 does so, comparing the performance of both planners under increasingly fine-grained levels of discretization. It finds the knapsack planner to perform more consistently than the greedy planner, and its peak performance to be higher.

This experiment also provides some information about the cost of discretization itself. Between $B = 1$ and $B \approx 60$, the success rate climbs from 0.62 to 0.67, corresponding to solving more than 40 additional instances on average. Coarse discretization does harm performance. The existence of the plateau in success rate above $B \approx 60$, however, shows that the number of bins required to escape this cost is not impracticably high. Furthermore, the relationship between discretization and performance can be surprising: while additional bins do tend to improve performance, the improvement is not monotonic. Since bins are evenly spaced, the addition of a new bin affects the locations of all bins, and can thus negatively affect the set of schedules available for consideration by a planner.

The planners may also be compared by examining, directly, the plans they compute for a small suite of solvers on different categories of instances. Figure 6.3 does so for the `ppfolio` solver suite on three categories of the 2011 SAT competition, and overall. Both planners leverage large-scale patterns in performance. Neither, for example, allocates any processor time to the SLS solver `TNM` on the category of highly-structured INDUSTRIAL instances. The greedy planner, however, appears less likely to schedule long solver runs. This phenomenon is consistent with our understanding of the algorithm. As planning be-

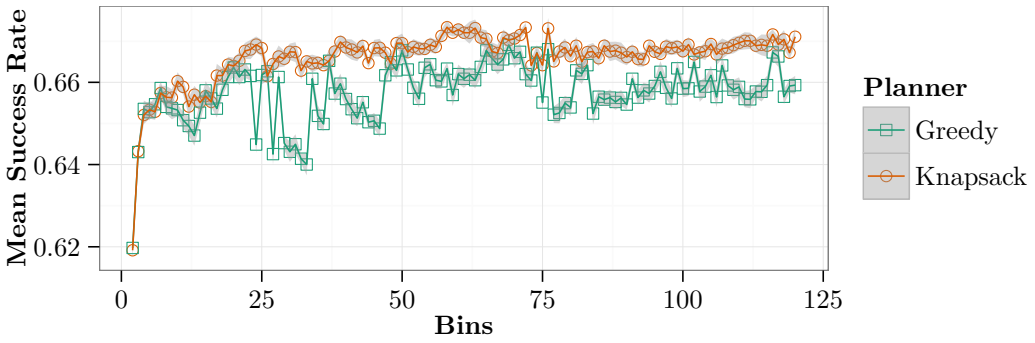


Figure 6.2: The mean fraction of all 2007 SAT competition instances solved by execution schedules for `SATzilla2009` subsolvers, computed across those instances by the greedy and knapsack planners, as the number of available solver run durations is increased. Results are averaged over solver nondeterminism. While both are approximate, the knapsack planner consistently outperforms the greedy planner.

gins, it may select an action that, in isolation, makes efficient use of processor time. Once selected, however, that action is locked into the plan, even though it may leave insufficient time for longer runs that, while less efficient, may be necessary to maximize the probability of solving an instance. The knapsack planner, while still likely to schedule short runs—especially of SLS solvers like `TNM`—on some categories, is also capable of dedicating its time to single long runs.

Figure 6.4 shows that this ability makes a difference in terms of the number of instances solved on these instance collections, and on the INDUSTRIAL category in particular. These plots highlight the connection between the number of instances solved and the amount of processor time required to solve them, exposing any tradeoff between speed and reliability. In every category, the shorter runs chosen by the greedy planner do solve slightly more instances quickly. In no category, however, does its plan solve more instances in total than does the knapsack planner, and its repeated `cryptominisat` runs solve many fewer

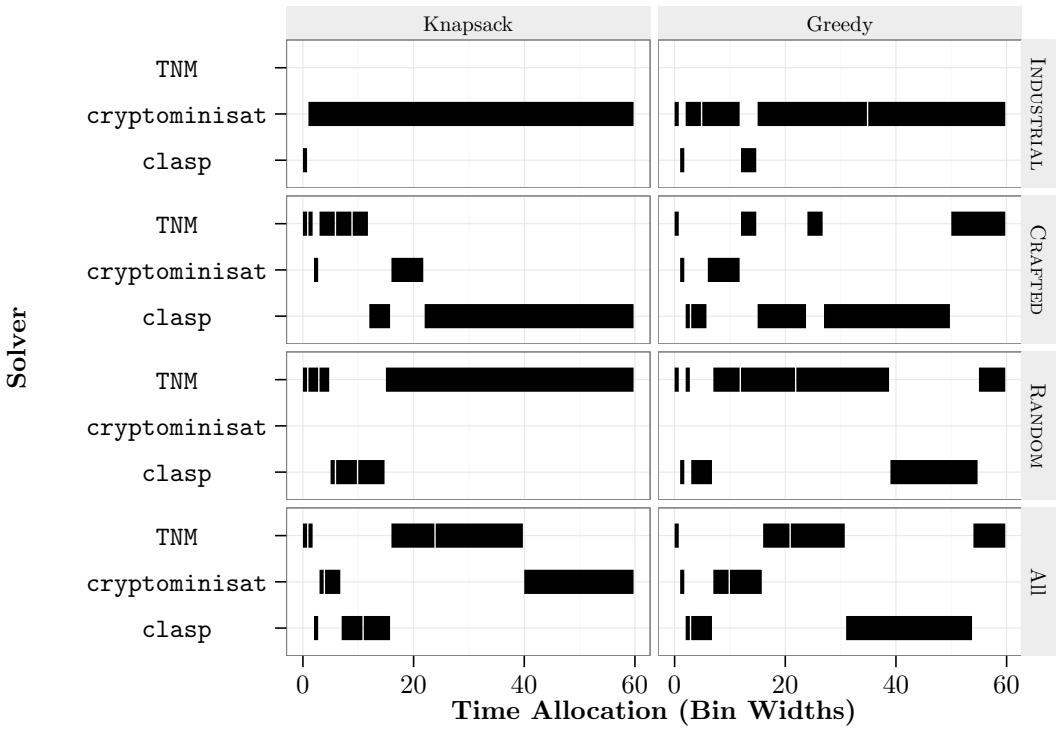


Figure 6.3: Plans computed by the greedy and knapsack planners for the `ppfolio` sub-solvers over all instances in each of SAT 2011 competition categories and overall. The effects of their algorithmic differences can be observed in the plans they compute. The greedy planner, which focuses on its actions' time-weighted probabilities of success, tends to make shorter runs. It is less likely to schedule long runs in certain domains; this effect is especially stark in the INDUSTRIAL category, on which the knapsack planner allocates the majority of its computation to a single `cryptominisat` run while the greedy planner makes repeated shorter runs. Figure 6.4 plots the number of instances solved by each plan, demonstrating that these differences affect performance.

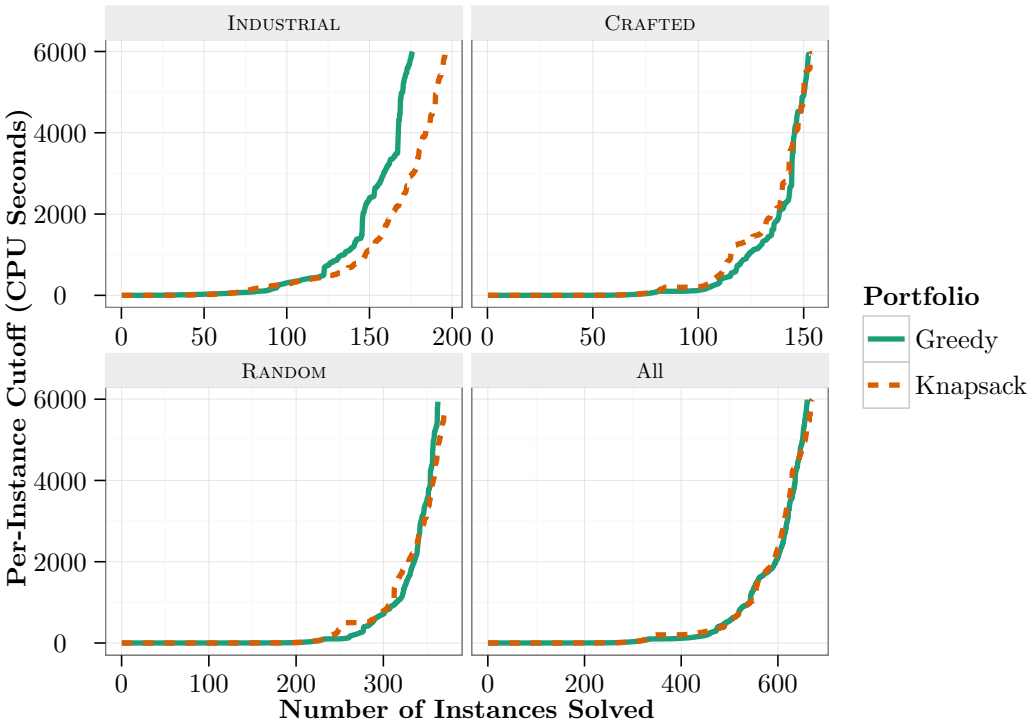


Figure 6.4: Cactus plots comparing the performance of the plans shown in Fig. 6.3 in terms of instances solved versus processor time required. The greedy planner tends to solve more instances quickly, a difference especially noticeable on CRAFTED, but it appears to do so at the cost of solving fewer overall. The knapsack planner instead successfully optimizes the number of instances solved. The longer runs in its schedules are, in fact, beneficial.

INDUSTRIAL instances than does the single long run of the knapsack plan. The dynamic programming approach appears to strike a good compromise between efficient and effective solver execution scheduling.

6.4 Conclusions

The planning algorithm presented in this chapter builds on a connection to the knapsack problem and employs dynamic programming for solver scheduling. In experiments on multiple categories, the plans computed by this approach outperform those of an existing greedy algorithm developed for the same purpose. Furthermore, the discretization it requires does not appear overly burdensome: on the collections of instances tested, it reaches its performance peak while the number of bins required remains reasonably small. The algorithm plays a critical role in MAPP, but also addresses the important need for simple, efficient solver scheduling in portfolio methods more generally.

The knapsack planning algorithm completes the third component of the MAPP framework. To solve a previously-unobserved test instance, the framework follows a solver execution schedule computed over its training instances; its performance model was previously used to estimate likely RTDs for its solvers on those instances. When domain-specific features are available, those RTDs are weighted according to the similarity-prediction scheme presented in Chapter 5, and the planner incorporates that weighting into its decisions. With the architecture complete, the next step is to compare the performance of this portfolio architecture with that of other methods. Chapter 7 will do so, presenting experiments that simulate past SAT competitions and pit MAPP directly against algorithm portfolios representing the state of the art.

Chapter 7

Evaluation in SAT

The individual components of MAPP were developed and evaluated in isolation in Chapters 4 to 6. They hold out the promise, however, of providing MAPP as a whole with compelling advantages over existing algorithm portfolio architectures. Its planning method is efficient and effective, allowing the portfolio to compute a schedule over many possible scenarios; it can choose to run nondeterministic solvers multiple times, employing them more effectively; its modeling step can handle nonstandard training sets and limited data; it can use instance features to consider a range of possible performance scenarios, while staying robust to irrelevant features. In practice, do these benefits translate to more instances solved?

This chapter measures and analyzes the performance of MAPP relative to that of other portfolio methods. Comparing directly against the three most recent portfolio systems to win in competition, it demonstrates that the conceptual benefits of MAPP are practical benefits as well. These evaluations are conducted by reenacting, approximately, the past solver competitions in which these alternative portfolios have been successful, and finding that MAPP indeed outperforms them.

	Benchmark Instances			
Year	**INDUSTRIAL**	**CRAFTED**	**RANDOM**	All
2011	300	300	600	1200
2009	290	281	610	1181
2007	175	201	511	887

Table 7.1: Number of instances in each category of the past three SAT competitions. The RANDOM category tends to include more instances than the others because arbitrary numbers of such instances can be generated. The "IBM" instances, which were not distributed in the main collection and on which few solvers were successful, are not included in the 2007 collection. That collection was used only for training. Knowledge of the sizes of these competitions allows the relative performance of each portfolio to be better understood.

7.1 Competition Benchmarks and Methodology

As discussed in Chapter 2, SAT solver competitions offer a valuable resource for judging the state of the field and for replicating experiments. Most importantly, they provide consistently packaged solver archives and a common set of benchmark collections. Major competitions are held every two years, and experiments in this chapter will use every instance from the 2007, 2009, and 2011 competitions. The sizes of these benchmark collections are summarized in Table 7.1, along with per-category breakdowns.

In competition, each solver is run on every instance in at least one of the three categories. In recent competitions, scoring has shifted from a purse-based scheme that rewarded multiple criteria (Van Gelder et al., 2005) to a simple ranking based solely on the number of instances solved, with ties broken by solver efficiency. These experiments—and the planner used by MAPP—focus on the number-of-solutions ranking, although efficiency will also be measured and discussed. Runs are subject to a standard CPU-time cutoff.

Competitions have used different cutoffs for different categories. The 2007 competition, for example, applied a 5,000 s cutoff to RANDOM and CRAFTED instances and a 10,000 s cutoff to INDUSTRIAL instances. In this chapter, for simplicity and consistency, and due to practical cluster constraints, a 6,000 s cutoff will be used in all categories. Four solver runs were collected on each training instance, and 60 performance bins were used for modeling and planning. A subset of the standard SAT features of Nudelman et al. (2004) is used to provide domain-specific instance information.

When they enter a solver competition, portfolio solvers are not granted prior access to the benchmarks on which they will be tested. Instead, they must train on some benchmark collection selected by their authors. This training set is often the collection of instances used by the previous competition. That scheme is applied in this chapter as well: each alternative portfolio was used exactly as submitted to the competition, trained on whatever set its authors chose, while MAPP was trained only on instances from the previous competition. Furthermore, comparisons against a competing portfolio were made using only its own suite of constituent solvers. In some cases, portfolios like SATzilla are trained on specific categories, and multiple specialized variants entered. Training of MAPP may, therefore, be focused on specific instance categories, but only as noted and only when the competing portfolio did so as well.

Despite this methodology, comparisons between portfolio methods are inherently inexact. Since experiments were run on a local cluster but competing portfolios were not retrained on that cluster, for example, any hardcoded timing information becomes less accurate in the local execution environment. These experiments can, therefore, only estimate the performance differences between methods, not quantify them precisely. Note that the same mismatch between training and test environments applies to portfolios executing in the official competition environment as well.

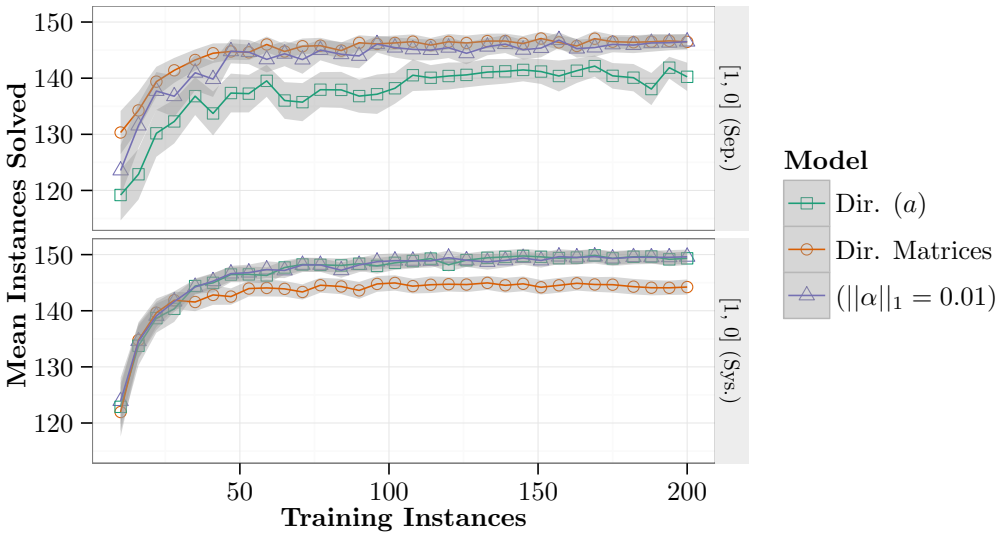


Figure 7.1: The mean number of 2009 competition instances solved by execution schedules for `ppfolio` subsolvers, computed using probabilities estimated under different models. Results were averaged over 128 random 80% train/test splits, varying the number of training instances used, and applying two different sampling strategies. Schedules computed with the simple model perform well when data for every solver are available on every instance ("[1, 0] (Sys.)"); when some data are missing ("[1, 0] (Sep.)"), the mixture models can infer the missing data, which results in better schedules.

7.2 Evaluating Model Utility

Models have thus far been compared by examining the probabilities they assign to held-out test data. These experiments provide a useful illustration of the models' differences, and such probabilities can be useful in their own right; for example, they can help understand or visualize solver behavior, as in the `borg-explorer` tool presented in Section 4.7. Figure 7.1 presents the outcome of an experiment that goes further: it demonstrates that a model's predictions can be valuable in the creation of a solver execution schedule. It also

demonstrates that the benefits of a complex model do not always outweigh the risk of being misled, and evaluates a way to mitigate this effect.

This experiment contrasts performance under two sets of solver run data collected in different ways. In the first set, all solvers were run systematically on the same subset of the training set. In the second set, solvers were run separately on different non-disjoint subsets. The same distinction was explored in the data-likelihood experiments of Fig. 4.5. As observed there, the appropriate model depends on the structure of available data. When plentiful runs have been systematically collected for training, as in the portfolio comparisons to follow, a uniformly-smoothed multinomial model is sufficient. It can also be quickly fit to data. For these reasons, it will be used in later comparisons.

When training runs have been sporadically collected, however, the predictable patterns in performance identified by the mixture model allow it to fill gaps in performance data—and the resulting execution schedules are more efficient. The results under systematic sampling show that the unconstrained mixture model can lead planning astray, with worse performance in that situation than under the smoothed multinomial model. To compensate for this weakness, the Dirichlet prior can be constrained ($||\alpha||_1 = 0.01$). This change allows observed run data to dominate when available, while inferring the likely distribution when data are unavailable. This combination enables the best schedules across sampling strategies.

In the next three sections, the performance of the entire MAPP approach is measured on standard benchmarks and compared with that of the three most successful portfolios in SAT to date: `SATzilla2009`, which dominated the 2009 competition (it did not compete in 2011); `ppfolio`, a simple parallel portfolio that performed well at the 2011 competition; and `3S`, a more complex portfolio using k-NN prediction that also performed well in 2011. Each of these portfolios won at least one competition category in 2009 or 2011. Results are illustrated through cactus plots that convey every portfolio's overall performance under the 6,000 s cutoff. Two other portfolios are included for perspective: a

"preplanning" portfolio, a stripped-down version of MAPP that applies the dynamic programming planner without the benefit of information provided by static instance features, and an oracle portfolio, which applies that planner to each instance given perfect knowledge of solvers' RTDs.

7.3 Comparison with `ppfolio`

The first comparison pits MAPP against a simple but successful naïve portfolio, `ppfolio`. This approach runs a manually-selected set of solvers in parallel, ignoring instance features and employing the trivial schedule in which every solver receives an equal share of the processor on every instance (Roussel, 2011). Despite its simplicity, it won more categories than any other solver that entered the 2011 competition. This dissertation evaluates the single-core version of `ppfolio`, in which only three solvers are used. Two of these solvers, `TNM` and `clasp`, are the winners of, respectively, the RANDOM and CRAFTED categories in the 2009 competition. The third solver, `cryptominisat`, won the smaller, applications-focused "SAT Race" in 2010. Although it can execute additional solvers on parallel hardware, such configurations are intended to optimize wall-clock rather than CPU time, and are not tested here.

Figure 7.2 compares the MAPP portfolios to `ppfolio` on all of the relevant competition categories, and overall. With feature information, MAPP outperforms `ppfolio` in every category. Its performance edge is greatest on RANDOM instances, a result that will be repeated in the comparisons to come. Since `ppfolio` is a simple parallel portfolio that ignores instance features, its performance can be matched—and, over all categories, slightly exceeded—by the preplanning version of MAPP as well. The plateaus evident in the performance of preplanning occur after a transition in its execution schedule, as the new solver quickly succeeds on the instances to which it is uniquely suited. Since `ppfolio` executes its solvers in parallel, it does not exhibit the same transition points. When MAPP is given access to feature information, its transition points also disappear: its schedules

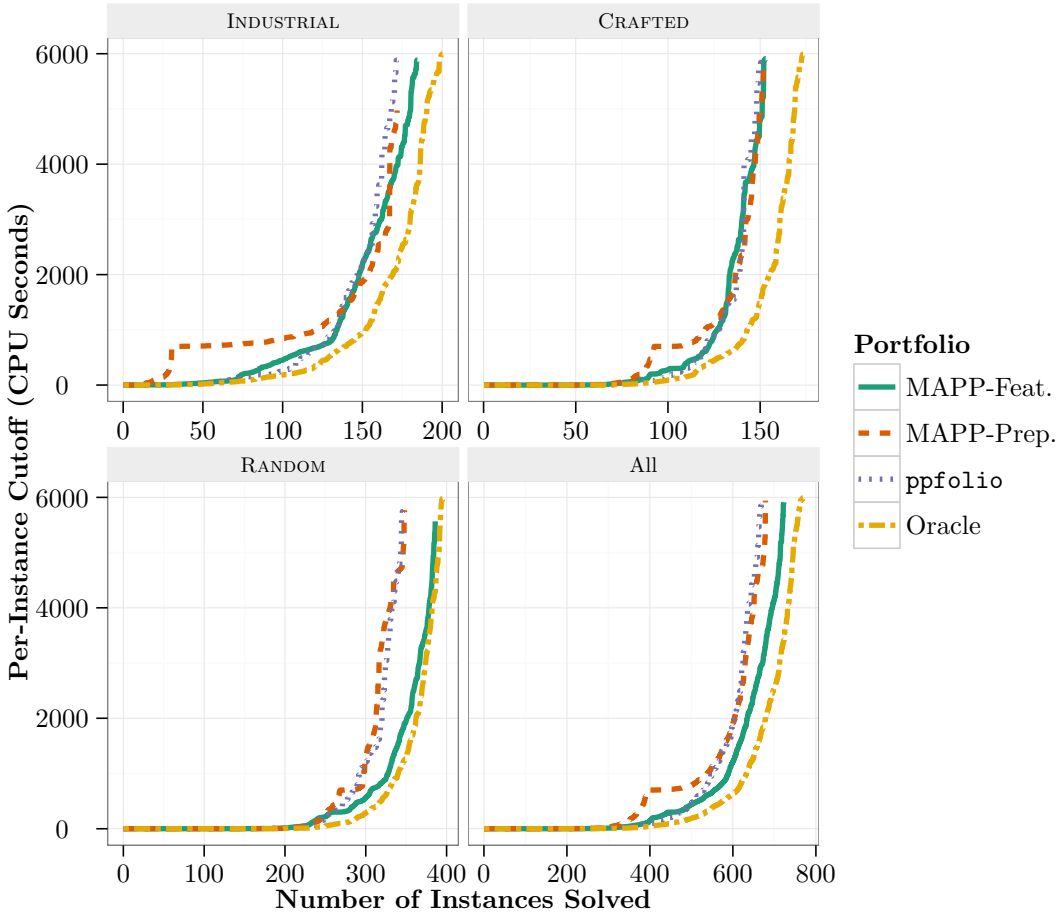


Figure 7.2: Cactus plots contrasting the mean performance of the MAPP portfolios with that of `ppfolio`. Performance overall, and on each of the three categories of the 2011 SAT competition, is shown averaged over solver and training nondeterminism. Portfolios were trained on all 2009 competition instances; they were not category-specific. Better performance curves are lower and farther to the right. With features, MAPP outperforms `ppfolio` on the INDUSTRIAL and RANDOM categories by a wide margin, and exceeds it slightly on CRAFTED. On RANDOM instances, its performance nearly reaches that of the oracle portfolio.

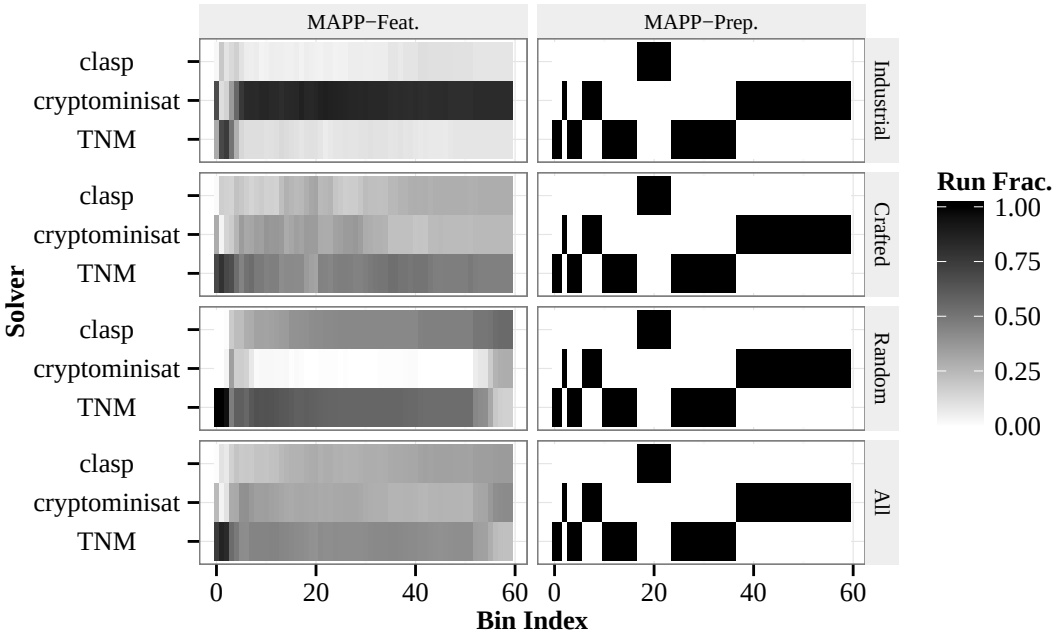


Figure 7.3: Summary of plans employed by the MAPP portfolios in the comparison presented in Fig. 7.2. The shading of each cell denotes the fraction of runs in which a particular solver was recorded as executing during a particular timeslice. Since the same preplanning portfolio is used in every category, its plans do not vary. With access to feature information, however, MAPP can predict the type of instance on which it runs, and tailor its schedule appropriately. This tailoring is evident in the schedules employed, such as the heavy use of `cryptominisat` on the INDUSTRIAL instances for which it was developed.

remain sequential, but their transitions occur at different times on different instances.

To peer more deeply into the behavior of MAPP, Fig. 7.3 visualizes logged records of the solver execution plans it followed, both when it did and not have access to feature information. With its early, short runs of `TNM`, MAPP has learned to employ a strategy similar to the "presolvers" scheme designed into SATzilla, in which a fixed set of several

solvers is run briefly on every instance. In general, the portfolio effectively tailors its run schedules to each category. Its plans on INDUSTRIAL instances are dominated by long runs of the CDCL solver `cryptominisat`. On RANDOM instances, a mixture of `TNM` and `clasp` runs is used, reflecting the split between satisfiable and unsatisfiable instances. On CRAFTED instances, a mixture of all three solvers reflects the heterogeneous nature of the category.

This comparison with `ppfolio` measured the performance of MAPP against a simple alternative portfolio architecture, and established that MAPP can perform well using a small suite of solvers. The next section gathers performance data in a different region of the space of portfolio environments: using a mid-sized solver suite, and measuring its performance relative to the standard SATzilla algorithm-selection architecture.

7.4 Comparison with `SATzilla2009`

The architecture of SATzilla, described further in Section 3.2.2, forms the basis of other algorithm-selection portfolios such as `claspfolio`, against which MAPP is compared in the next chapter. The fundamental principle of SATzilla is the use of instance features to select a single solver, which is then run to completion on a given instance. Secondary functionality includes a small set of presolvers run briefly on every instance, and a fixed backup solver run only if the selected solver terminates prematurely. `SATzilla2009` is the most recent version of this architecture to enter a SAT competition.

Figure 7.4 shows that, despite using fewer features, MAPP performs substantially better than `SATzilla2009` on RANDOM instances, matches it on INDUSTRIAL, and slightly exceeds it on CRAFTED. On RANDOM, the planner appears to trade off faster solutions to easier instances for attainable solutions to more difficult ones. While the performance curve of the preplanning portfolio is worse overall, it is both interesting and surprising to observe it matching the overall success rate of `SATzilla2009` on that category even though it does not apply domain-specific information. On this category, the combination of restarts

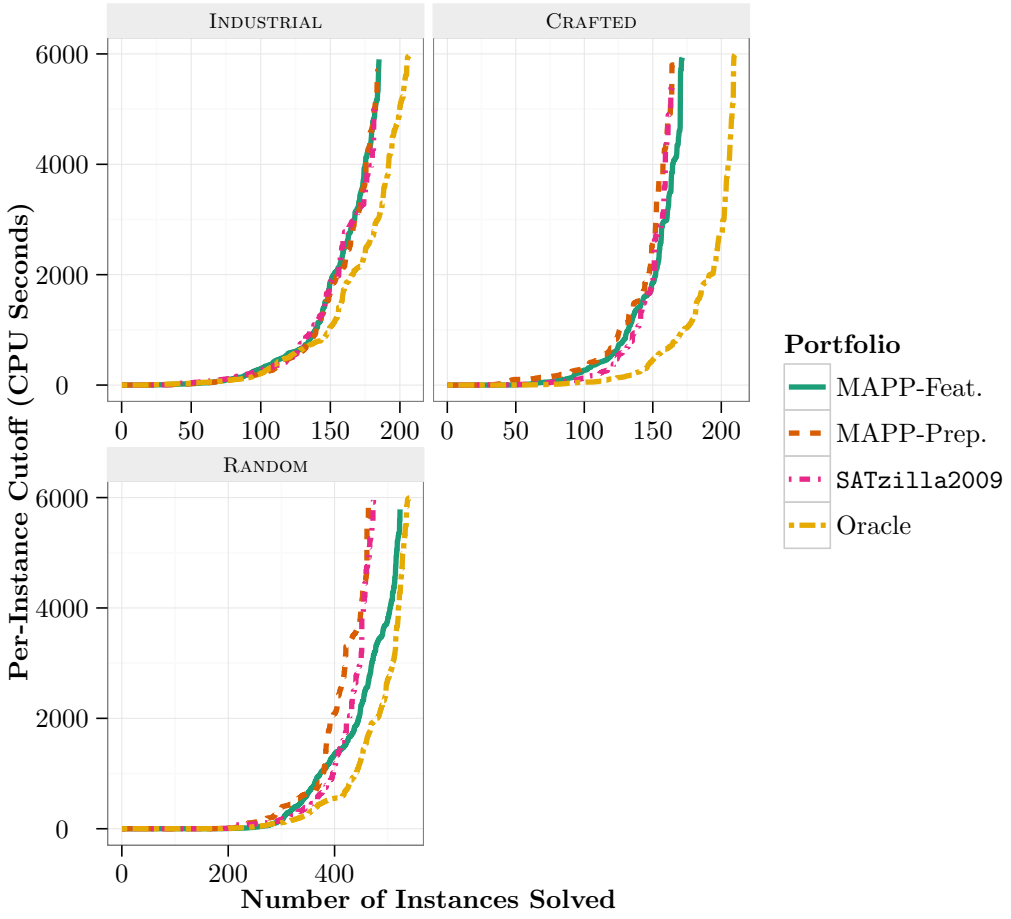


Figure 7.4: Cactus plots contrasting the mean performance of MAPP with that of SATzilla2009. Because no version of SATzilla entered the 2011 competition, 2007 instances were used for training and 2009 for testing. Furthermore, since three category-specific variants of SATzilla2009 were entered, the other portfolios were also trained and tested separately on each category, and no all-category experiment could be conducted. With features, MAPP matches or exceeds the performance of SATzilla2009 in every case. On RANDOM, it solves fewer instances in less than 1,000 s, but many more instances overall.

and multiple solvers appears to be particularly valuable. The same result was suggested by the earlier experiments of Silverthorn and Miikkulainen (2010), which focused on modeling "burstiness", i.e., the degree to which restarts affect solver success.

Table 7.2: Instance features used by "MAPP-Feat." in the experiments in this chapter. This feature set is a strict subset of that used by `3S` and by `SATzilla2009`, which computes up to 96 features (Xu et al., 2009), but observed performance shows that it provides sufficient information to MAPP.

Name	**Description**
`BINARY+`	Number of unary and binary clauses.
`horn-clauses-fraction`	Fraction of clauses that are Horn clauses.
`HORNY-VAR-cv`	CV of the variables' Horn clause counts.
`HORNY-VAR-entropy`	Entropy of variables' Horn clause counts.
`HORNY-VAR-max`	Max. number of Horn clauses in which a variable appears.
`HORNY-VAR-mean`	Mean number of Horn clauses in which a variable appears.
`HORNY-VAR-min`	Min. number of Horn clauses in which a variable appears.
`nclauses`	Number of clauses.
`nvars`	Number of variables.
`PN-RATIO-CLAUSE-cv`	CV of the ratios of pos. to neg. literals in clauses.
`PN-RATIO-CLAUSE-entropy`	Entropy of the ratios of pos. to neg. literals in clauses.
`PN-RATIO-CLAUSE-max`	Maximum ratio of positive to negative literals in a clause.
`PN-RATIO-CLAUSE-mean`	Mean ratio of positive to negative literals in a clause.
`PN-RATIO-CLAUSE-min`	Minimum ratio of positive to negative literals in a clause.
`PN-RATIO-VAR-entropy`	Entropy of the ratios of pos. to neg. literals for variables.
`PN-RATIO-VAR-max`	Maximum ratio of pos. to neg. literals for a variable.
`PN-RATIO-VAR-mean`	Mean ratio of positive to negative literals for a variable.
`PN-RATIO-VAR-min`	Minimum ratio of pos. to neg. literals for a variable.

Table 7.2: Continued.

PN-RATIO-VAR-stdev	Std. dev. of ratios of pos. to neg. literals for variables.
TRINARY+	Number of unary, binary, and trinary clauses.
UNARY	Number of unit clauses.
vars-clauses-ratio	Ratio of variables to clauses.
VCG-CLAUSE-cv	CV of clause-node degrees in variable-clause graph.
VCG-CLAUSE-entropy	Entropy of clause-node degrees in variable-clause graph.
VCG-CLAUSE-max	Maximum clause-node degree in variable-clause graph.
VCG-CLAUSE-mean	Mean clause-node degree in variable-clause graph.
VCG-CLAUSE-min	Minimum clause-node degree in variable-clause graph.
VCG-VAR-cv	CV of variable-node degrees in variable-clause graph.
VCG-VAR-entropy	Entropy of variable-node degrees in variable-clause graph.
VCG-VAR-max	Maximum clause-node degree in variable-clause graph.
VCG-VAR-mean	Mean clause-node degree in variable-clause graph.
VCG-VAR-min	Minimum clause-node degree in variable-clause graph.
VG-cv	CV of node degrees in the variable graph.
VG-max	Maximum node degree in the variable graph.
VG-mean	Mean node degree in the variable graph.
VG-min	Minimum node degree in the variable graph.

The feature set used by MAPP, listed in Table 7.2, is a small subset of that used by SATzilla2009. A smaller set provides less information, but obviates the need for the more complex scheme employed by SATzilla2009. That scheme trains a regression model to predict the cost of feature computation, and avoids it if that cost is likely to exceed a specified threshold (Xu et al., 2009). The success of MAPP shows that the information provided by the larger feature set is not essential for competitive performance, perhaps because solver scheduling can partly compensate for the additional uncertainty.

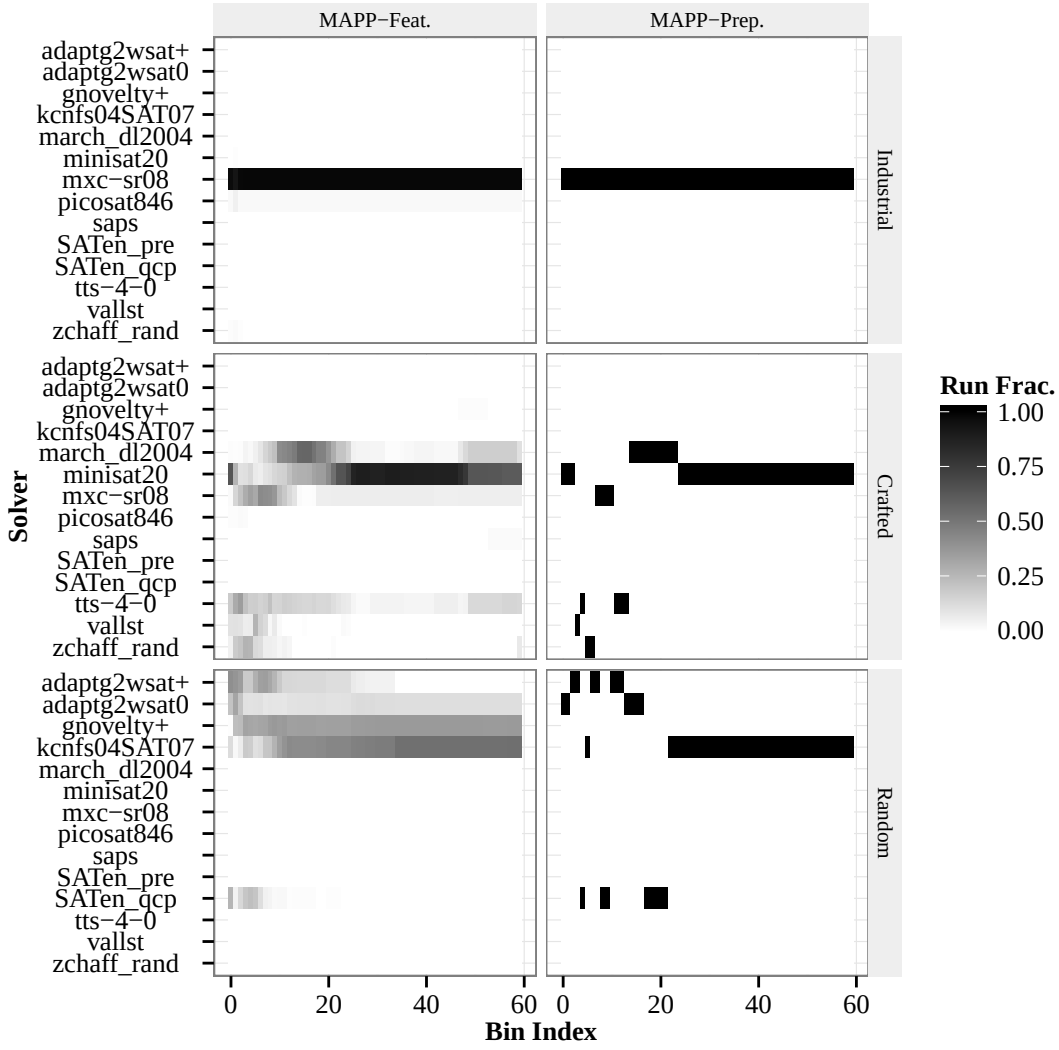


Figure 7.5: The fraction of runs in which MAPP executed each solver in each bin, in runs on 2011 instances using the `SATzilla2009` solver suite. Since, like `SATzilla2009`, a specialized portfolio was trained on each category, multiple plans are computed by the preplanning portfolio. Both with and without features, plans differ strongly between categories, ranging from an essentially single-solver strategy on INDUSTRIAL—although `picosat846` is, in fact, executed on a small fraction of instances—to a blended strategy with multiple restarts on RANDOM. This range illustrates the flexibility of the MAPP approach.

The plans employed by the MAPP portfolios are presented in Fig. 7.5. The plans computed by portfolios trained per-category, as in this section, contrast starkly with those computed by a single general-purpose portfolio, as in Section 7.3. Portfolios in this section generate plans that focus exclusively on a subset of solvers, and the sets of solvers executed on each category are almost disjoint. In some cases, such as the INDUSTRIAL category, the portfolio appears to employ a single solver almost entirely—mirroring, in a different solver suite, the results of Fig. 6.3, and explaining all portfolios' similar performance on INDUSTRIAL instances in Fig. 7.4. The contrast between these plans shows that instance features are unable to distinguish between categories of instances with certainty, at least given the supervised learning methods employed. The plans also expose another point in favor of portfolio methods: solvers that would otherwise be considered obsolete—such as `zchaff_rand`—still play a role in the portfolio. Such solvers are unlikely to be considered by a human user.

The flexibility of MAPP, especially the support for overall solver scheduling, seems justified by this comparison with an algorithm-selection scheme. The final comparison will pit MAPP against a competing portfolio that uses a larger suite of solvers, and that, like MAPP, computes solver schedules—but without restarts, and without MAPP's more sophisticated feature integration. The comparison will thus measure the advantages conferred by these components specifically.

7.5 Comparison with `3S`

The "satisfiability solver selector", `3S`, is a new approach entered in the 2011 competition. Unlike the algorithm-selection architecture of `SATzilla2009`, `3S` computes execution schedules: for each test instance, it follows a schedule tailored to that instance's nearest neighbors in feature space. Its set of features is a subset of those established by `SATzilla`, and larger than that implemented in MAPP. Unlike the simple, efficient planner used by MAPP, `3S` uses a general-purpose IP solver and plans only over a small number

of instances.

Although both portfolios use planning, Fig. 7.6 shows that their performance differs markedly. With features, MAPP measurably outperforms `3S` in every category of the 2011 competition. Their joint performance advantage over preplanning is more pronounced in this context than in other two comparisons, especially on INDUSTRIAL instances. Two reasons may cause this difference: unlike in Fig. 7.4, the portfolios are not category-specific, and, unlike in Fig. 7.2, a large solver suite with correspondingly greater performance diversity is used.

As in previous comparisons, Figs. 7.7 to 7.10 visualize the plans used by MAPP. The combination of a shared training set and a large solver suite apparently causes a wider variety of solvers to be used, but obvious differences remain between the plans applied to instances in different categories. Unlike in previous comparisons, multiple solvers are employed on the INDUSTRIAL category: both `lingeling` and `cryptominisat` receive significant CPU time. On RANDOM, time is largely split between SLS solvers (`TNM`, `hybridGM3`, etc.) and a single CDCL solver (`march_hi`), as in every previous effective schedule on such instances. Even in this challenging evaluation, in other words, trained on multiple categories of a different competition, using a large solver suite, MAPP continues to execute good, category-specific solver execution plans.

7.6 Conclusions

In all experiments, MAPP solves at least as many instances as the competing portfolio in every category, and solves measurably more instances in at least one category. These results suggest that the framework presented in this work improves upon existing portfolio methods, while employing a novel combination of techniques.

This picture of MAPP performance illustrates its comparative strengths. It provides the greatest advantage, over both single solvers and existing portfolio methods, on heterogeneous instance sets, such as a mix of satisfiable and unsatisfiable instances of random

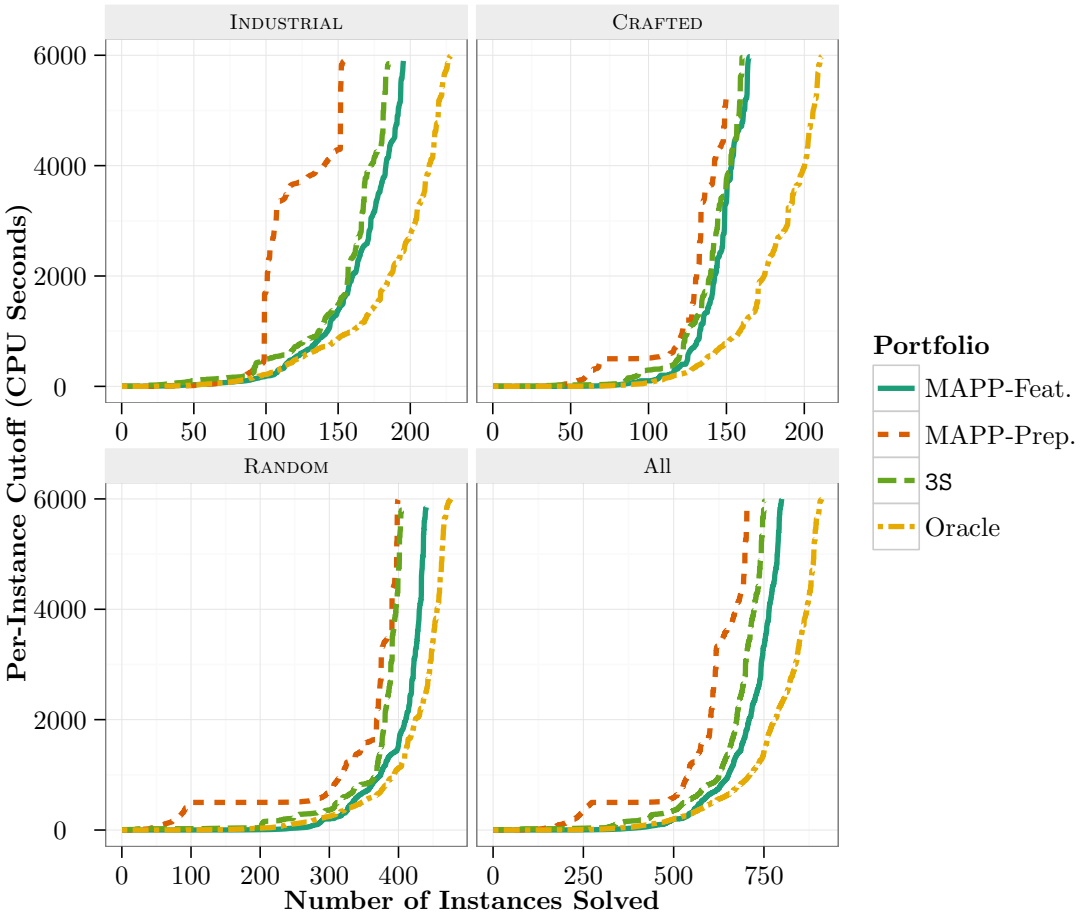


Figure 7.6: Cactus plots contrasting the mean performance of MAPP with that of 3S in the 2011 SAT competition environment. Portfolios were not category-specific. The probabilistic portfolio exceeds the performance of 3S in every category, although the difference on CRAFTED is small. The gap versus oracle is large throughout, perhaps due to the large size of the 3S solver suite.

k-SAT, and when using a suite of substantially nondeterministic solvers, such as those in the SLS family. In these situations, the combination of complete and incomplete solvers allows the portfolio to handle both SAT and UNSAT instances, while MAPP's integrated restart scheduling maximizes the value of stochastic search strategies. This strength can be seen in the consistently strong performance of the MAPP portfolios on the RANDOM category.

These results also illuminate both the power and the limits of portfolio methods in general. In categories where long runs are required and divisions between instances less clear, such as on the collections of INDUSTRIAL instances in Section 7.3 and Section 7.4, the performance of all portfolios can converge as their strategies collapse to a single solver run. Note, however, that—tested on the same set of instances—what was a single-solver strategy in Section 7.4 becomes an informed mix of multiple solvers in Section 7.5, as the suite of solver options expands and training data become more ambiguous. The promise of portfolio methods, of leveraging empirical data to make the often-subtle judgments between alternative algorithms, seems justified by these data: it is better to have a portfolio method decide whether to apply one or many algorithms, than to make that difficult judgment oneself. The next chapter will move these experiments beyond SAT, applying MAPP to PB, MAX-SAT, and ASP. This work will test whether the effectiveness of portfolios in general, and of MAPP in particular, extends to new domains.

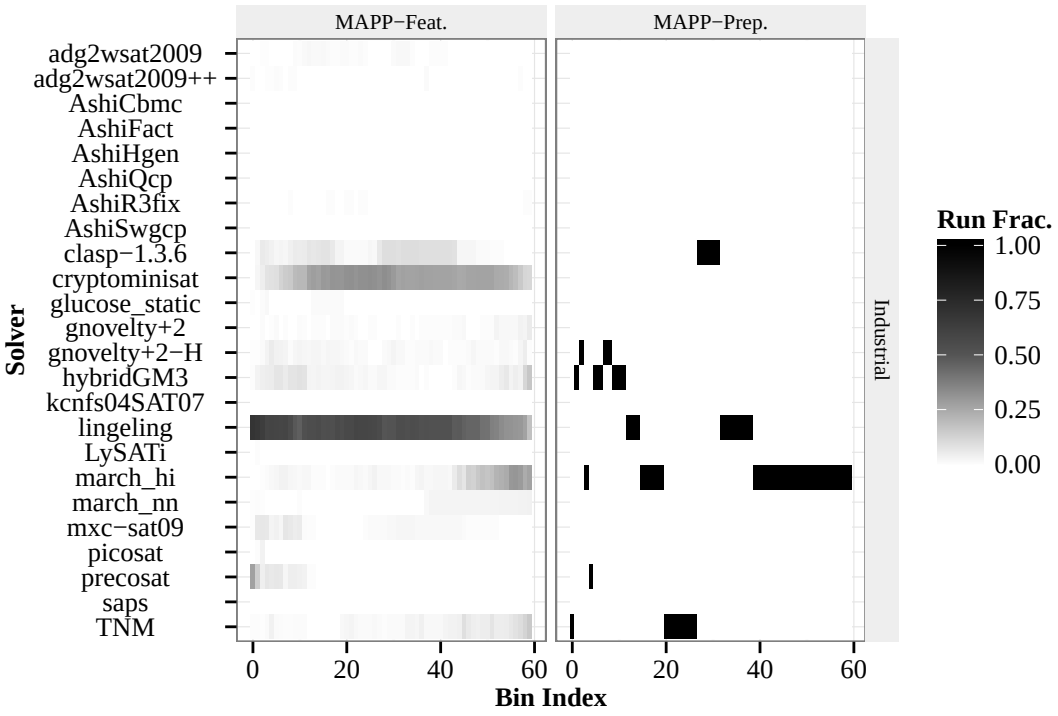


Figure 7.7: Solver executions, averaged over instances in the INDUSTRIAL category, selected by the MAPP portfolios in the comparison with `3S`. The shared training set appears to make plans less rigid—even SLS solvers such as `hybridGM3` are occasionally run on INDUSTRIAL instances—but distinctions between the strategies on each category are apparent when comparing this figure to Figs. 7.8 to 7.10.

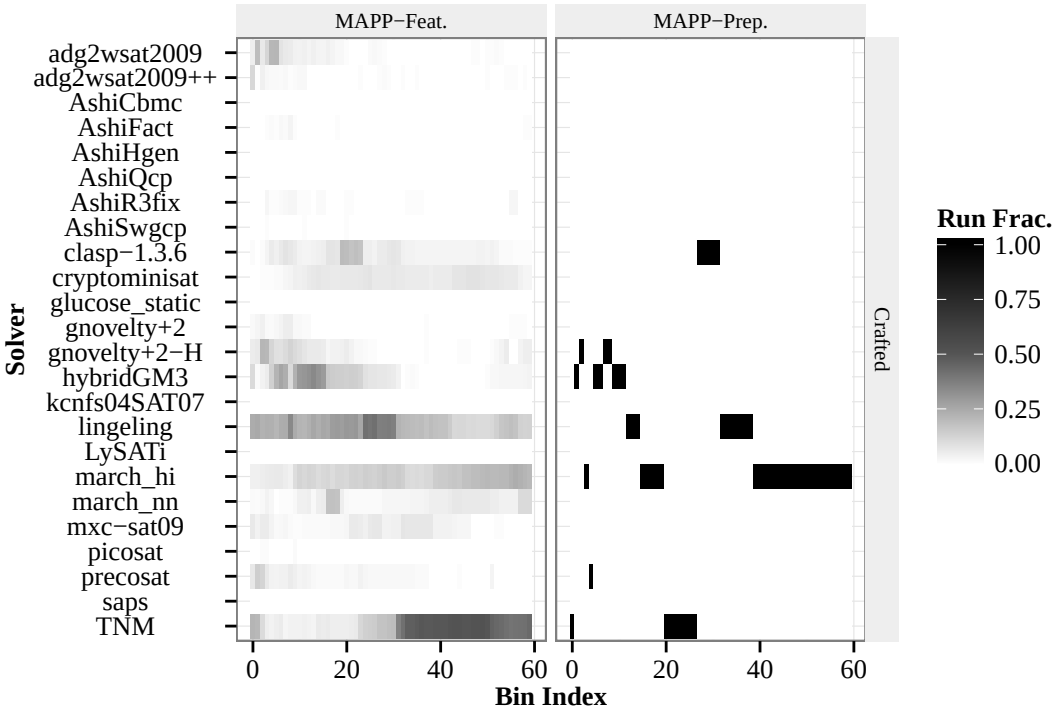


Figure 7.8: The plan visualizations of MAPP in the 3S comparison, continued; this figure presents average solver executions in the CRAFTED category.

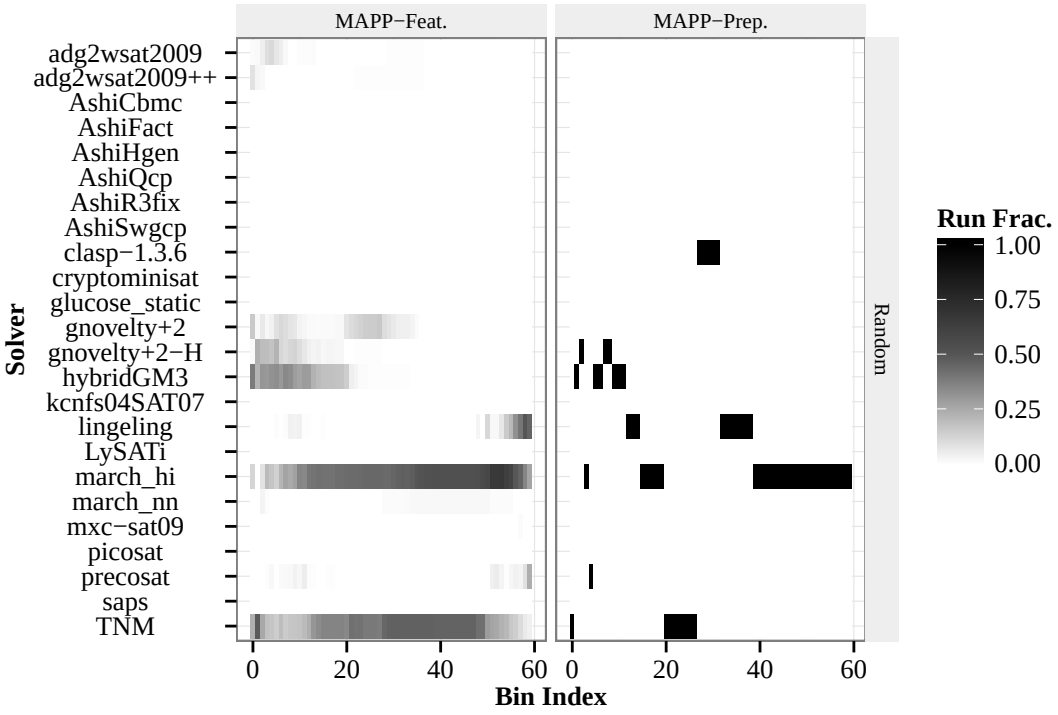


Figure 7.9: The plan visualizations of MAPP in the 3S comparison, continued; this figure presents average solver executions in the RANDOM category.

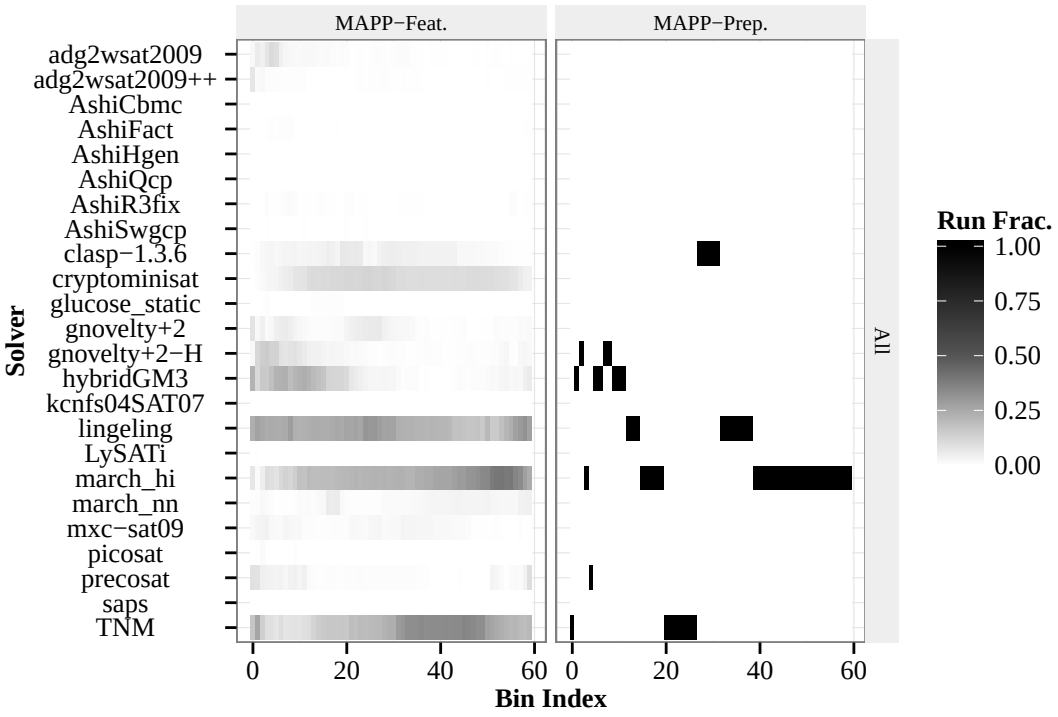


Figure 7.10: The plan visualizations of Figs. 7.7 to 7.9, continued; this figure presents average solver executions across all categories. Considering these figures together, the set of solvers applied to RANDOM instances overlaps with that for CRAFTED instances, but different solvers are emphasized—`lingeling`, for example, a heavyweight CDCL solver suited to large, structured instances but ill-suited to random k-SAT, is run earlier, longer, and more frequently on CRAFTED. MAPP continues to focus on effective strategies even with a large solver suite and even without category-specific training.

Chapter 8

Evaluations in PB, MAX-SAT, and ASP

Existing portfolio approaches are often constructed to target specific domains. `SATzilla`, for example, was built and named with a focus squarely on SAT. In contrast, the probabilistic portfolio framework presented in this dissertation was developed specifically to be more versatile. To evaluate whether that goal was achieved, this chapter applies MAPP to three domains beyond pure SAT: to pseudo-Boolean satisfiability, in Section 8.1; to maximum satisfiability, in Section 8.2; and to answer set programming, in Section 8.3. Without altering its core configuration, MAPP outperforms its constituent solvers in all of these domains, measurably improves its performance by leveraging domain-specific features when they are provided, and compares well to a domain-specific portfolio even when it is limited to deterministic solvers. These results confirm that its original design goal was reached: MAPP makes effective algorithm scheduling decisions in multiple domains.

8.1 Comparisons in Pseudo-Boolean Satisfiability

Evaluations in these new domains are similar to those of earlier sections: domain-specific instance features are identified, an appropriate suite of solvers is compiled, runs of those solvers are collected on training instances, and the performance characteristics of different solvers and portfolios are measured and compared on test instances.

The first new domain to be examined, pseudo-Boolean satisfiability (PB), replaces the propositional formulae of SAT with more general linear inequality constraints on integer variables in $\{0, 1\}$. A PB instance may optionally include an optimization objective, and a further generalization supports nonlinear constraints. PB was described in greater detail by Section 2.3.

No other prominent portfolio exists for PB. Experiments in this domain therefore focus on measuring the improvement offered by the probabilistic portfolios over individual solvers, and on measuring the performance advantage, if any, gained by access to instance feature information. Instances from the 2010 PB competition are used, under cross-validation, both for training and testing. Table 8.2 lists the solvers included in the portfolio. Solvers in PB are expected to assert the optimality of solutions to optimization instances, so they are Las Vegas algorithms that can be handled, even for such instances, in exactly the same way as solvers in SAT.

Table 8.1: The 20 instance features applied to the PB domain, ordered according to their relevance in RTD-similarity classification across all categories. Relevance is measured by the mean magnitudes of the corresponding classifier coefficients, after each feature is scaled to zero mean and unit variance. This set of features is small and straightforward, but its use improves portfolio decisions, confirming that significant feature engineering is not necessarily required.

Weight	Name	Description

Table 8.1: Continued.

1.26	totals_log_max	Log of the absolute value of the largest-magnitude sum used in a constraint.
1.24	totals_log_std	Log of the standard deviation of the sums used by constraints.
1.18	variables	Number of variables used.
1.06	vcg_cnode_deg_std	Standard dev. of clause-node degrees in the var/-clause graph.
1.05	ratio_reciprocal	Reciprocal of the constraints / variables ratio.
1.02	vcg_cnode_deg_mean	Mean clause-node degree in the var/clause graph.
1.01	vcg_vnode_deg_std	Standard dev. of var-node degrees in the var/-clause graph.
0.95	optimization	Does the instance specify an objective?
0.95	ratio	Ratio of constraints to variables.
0.90	vg_node_deg_std	Standard deviation of node degrees in the variable graph.
0.89	totals_log_min	Log of the absolute value of the smallest-magnitude sum used in a constraint.
0.86	cg_cnode_deg_std	Standard deviation of node degrees in the clause graph.
0.77	totals_log_mean	Log of the absolute value of the mean sum across all constraints.
0.77	vcg_vnode_deg_mean	Mean var-node degree in the var/clause graph.
0.74	constraints	Number of inequality constraints used.
0.69	nonlinear	Is the instance nonlinear?
0.58	cg_cnode_deg_mean	Mean node degree in the clause graph.
0.53	vg_node_deg_mean	Mean node degree in the variable graph.

Table 8.1: Continued.

0.47	`coef_means_std`	Standard deviation of constraint coefficient means.
0.23	`coef_means_mean`	Mean of constraint coefficient means.

To measure their value, a modest set of static features is assembled. These features are described in Table 8.1. Some, such as the statistics of the clause, variable, and clause-variable graphs, mirror those developed by Nudelman et al. (2004) for CNF SAT. Others, such as the statistics of clause coefficients, are domain-specific. The importance of each feature is roughly estimated by its mean classifier coefficient weight. This ranking shows that PB-specific features, especially log-scale statistics of the inequality thresholds, are particularly useful, but that certain standard SAT features also remain valuable.

Cactus plots of portfolio performance on competition benchmarks in PB are presented in Fig. 8.1. The individual solver that performs best on training instances is taken as a baseline. The probabilistic portfolios, both with and without access to feature information, solve substantially more instances than this baseline in both the decision and optimization categories. Feature information, however, further improves efficiency.

In addition to the results of these local experiments, precursors to MAPP entered and won the decision-problem category of both the 2010 and 2011 PB competitions, under the system name `borg`. Table 8.3 lists the results of the most recent competition. Portfolios entering such competitions can only include constituent solvers available to them well before the competition, and must therefore demonstrate their effectiveness against a larger, more advanced field of competitors. The success of the MAPP approach in the competition setting demonstrates that it can operate well even outside of the controlled experimental environment in this dissertation, i.e., in a rigorous, externally-administered comparison. It also shows that the advantages conferred by a portfolio strategy can, in some situations, outweigh disadvantages in the composition of its solver suite.

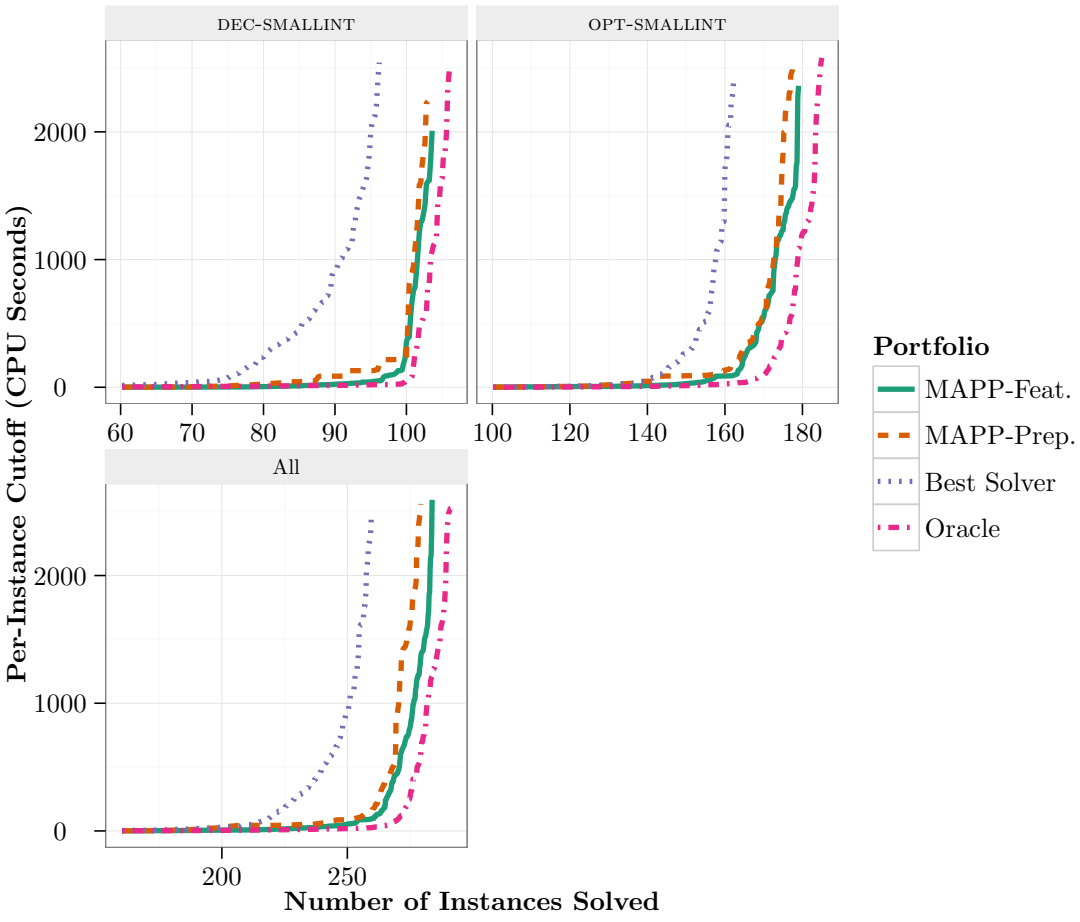


Figure 8.1: Cactus plots comparing the performance of probabilistic portfolios in PB, with and without using static instance features, to oracle performance as well as the single-solver baseline. Performance averages were computed over five-fold cross validation on the 2010 PB competition instances. As it did in pure SAT, MAPP dramatically outperforms individual solvers. Furthermore, the use of static instance features measurably improves solving efficiency versus pure preplanning.

Name	Author(s)
pbct-0.1.2-linear	Anders Franzén and Roberto Bruttomesso
sat4j-pb-v20101225	Daniel Le Berre and Anne Parrain
sat4j-pb-v20101225-cutting	Daniel Le Berre and Anne Parrain
bsolo_pb10-l1	Vasco Manquinho and José Santos
bsolo_pb10-l2	Vasco Manquinho and José Santos
bsolo_pb10-l3	Vasco Manquinho and José Santos
wbo1.4a	Vasco Manquinho, Joao-Marques Silva, and Jordi Planes
wbo1.4b-fixed	Vasco Manquinho, Joao-Marques Silva, and Jordi Planes
clasp-1.3.7	Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub
scip-2.0.1-clp	Stefan Heinz and Michael Winkler
scip-2.0.1-spx	Stefan Heinz and Michael Winkler

Table 8.2: The solvers included in the PB portfolio evaluated in this section. The range of solvers is smaller in PB than in SAT, and this portfolio includes multiple parameterizations of certain solvers. The results of Fig. 8.1, however, confirm that even this smaller portfolio significantly outperforms in combination any one of its members in isolation.

Next, the same methodology is applied to a domain that generalizes propositional SAT in a different direction.

8.2 Comparisons in Maximum Satisfiability

PB replaces the Boolean constraints of SAT with inequalities. The problem of maximum satisfiability (MAX-SAT) instead maintains the structure of CNF SAT, but expands the definition of an acceptable solution, allowing certain clauses to be left unsatisfied if necessary.

		Solved			
Rank	**Solver**	**Total**	**SAT**	**UNSAT**	**Cum. CPU (s)**
1	borg-pb-dec	431	183	248	10281.04
2	Sat4j-Res//CP	420	183	237	51588.82
3	bsolo-3.2	416	179	237	58921.79
4	wbo-1.6	394	180	214	17024.67
5	Sat4j-Resolution	392	184	208	33537.21
6	SCIP-spx-E_2	384	149	235	46478.58
7	SCIP-spx-2	383	148	235	44742.45
8	clasp-2.0-R4191	380	168	212	12375.19
9	MinisatID-2.5.2	368	169	199	19281.13
10	MinisatID-2.5.2-gmp	362	165	197	37474.45
11	Sat4j-CuttingPlanes	242	114	128	22807.02

Table 8.3: Results of the main decision-problem category (DEC-SMALLINT) of the 2011 PB solver competition. As in the 2010 competition, the winner was a precursor of MAPP entered under the system name borg-pb-dec. It solved many more instances than did the second-place solver, while using less than 20% of the total CPU time. The overall difference in efficiency is striking: despite solving the greatest number of instances, the probabilistic portfolio accumulated the lowest *total* CPU time of any solver over completed instances. This success appears due to MAPP effectively employing both SAT-derived solvers based on resolution and IP-derived solvers based on cutting-plane inference. The competition thus provides an example of MAPP automatically allocating computational resources across solution strategies whose strengths may be difficult to integrate into a single solver.

MAX-SAT solvers, like those for PB, are expected to assert optimality, and are therefore Las Vegas algorithms suitable for inclusion in the probabilistic portfolio framework. The MAX-SAT problem was described in detail by Section 2.2. The methodology of this section is identical to that of Section 8.1: solvers and portfolios are evaluated, under cross validation, on the instances used by a recent competition.

The suite of solvers included in this portfolio, listed in Table 8.4, is even smaller than that in PB. Nonetheless, Fig. 8.2 shows that the probabilistic portfolio solves more instances, more quickly, both overall and in the weighted and unweighted partial MAX-SAT categories (PMS and WPMS), than the best single solver. In the two categories in which the portfolio architecture offers little improvement (MS and WMS), the gap between that baseline and oracle performance is small: MAPP is unhelpful because, like any portfolio architecture, it relies on substantial performance diversity among its constituent solvers. On these categories, the solvers themselves are not complementary.

In those categories where such diversity exists, domain-specific features are again successfully leveraged. Table 8.5 describes these features. Furthermore, the breadth of experiments in this chapter allows the value of analogous features in different domains to be compared. Differences are evident between the value of such features in PB, as reported in Table 8.1, and in MAX-SAT. While some features (e.g., `ratio_reciprocal`) appear informative in both domains, others (e.g., `variables` and `vcg_vnode_degree_mean`) are weighted heavily in one but not the other. This fluctuation suggests that features should be pruned with caution, if at all: their value may depend unpredictably on the domain and the instance distribution.

Two other conclusions are clear from the MAX-SAT domain. First, a portfolio can provide substantial benefit even when few solvers are available. Second, if solvers do not offer complementary strengths to be leveraged for some distribution of instances, a portfolio method can track both oracle and best-single-solver performance. Thus, if a portfolio method avoids expensive instance features, its application need not incur unnecessary cost.

Name	Author(s)
akmaxsat	Adrian Kuegel
sat4j-maxsat-v20101225	Daniel Le Berre
sat4j-maxsat-v20101225-p	Daniel Le Berre
inc-maxsatz	Han Lin, Kaile Su, and Chu Min Li
IncWMaxSatz	Han Lin, Kaile Su, Chu Min Li, and Josep Argelich
wbo1.4b-fixed	Vasco Manquinho, Joao-Marques Silva, and Jordi Planes

Table 8.4: The six MAX-SAT solvers used in the portfolio evaluation presented by Fig. 8.2. The IncWMaxSatz and inc-maxsatz solvers are applied exclusively to weighted and unweighted instances respectively; only five distinct choices are therefore available. This small collection of solvers has too little performance diversity to allow a portfolio to provide benefit in every category (i.e., one solver largely dominates on MS and WMS instances). Its solvers are, however, sufficiently complementary on multiple other categories.

These evaluations in PB and MAX-SAT have measured the advantage over individual solvers produced by the probabilistic portfolio approach. The third evaluation beyond SAT, in the domain of ASP, will include another portfolio method in the comparison.

Table 8.5: The 20 static instance features used in the MAX-SAT domain, ordered according to their estimated importance to nearest-RTD classification. Other than the clause weight statistics, the features are a subset of those used in pure CNF SAT. As in PB, even this modest feature set allows the portfolio to allocate processor time better.

Weight	Name	Description

Table 8.5: Continued.

1.51	ratio	Ratio of constraints to variables.
1.49	vcg_vnode_deg_mean	Mean variable-node degree in the variable/clause graph.
1.10	vcg_vnode_deg_std	Standard deviation of variable-node degrees in the variable/clause graph.
0.97	constraints	Number of clauses in the Boolean formula.
0.97	ratio_reciprocal	Reciprocal of the constraints / variables ratio.
0.95	vcg_cnode_deg_mean	Mean clause-node degree in the variable/clause graph.
0.86	vcg_cnode_deg_std	Standard deviation of clause-node degrees in the variable/clause graph.
0.82	cg_cnode_deg_std	Standard deviation of node degrees in the clause graph.
0.81	var_bal_ratio_std	Standard deviation of variables' positive-/negative-literal ratios.
0.74	constr_bal_ratio_std	Standard deviation of clauses' positive-/negative-literal ratios.
0.69	constr_bal_ratio_mean	Mean of clauses' positive-/negative-literal ratios.
0.56	var_bal_ratio_mean	Mean of variables' positive-/negative-literal ratios.
0.54	cg_cnode_deg_mean	Mean node degree in the clause graph.
0.54	variables	Number of variables in the Boolean formula.

Table 8.5: Continued.

0.53	vg_node_deg_mean	Mean node degree in the variable graph.
0.48	vg_node_deg_std	Standard deviation of node degrees in the variable graph.
0.41	weights_min	Smallest clause weight to appear.
0.20	weights_max	Largest clause weight to appear.
0.20	weights_mean	Mean clause weight.
0.20	weights_std	Standard deviation of clause weights.

8.3 Comparisons in Answer Set Programming

MAX-SAT, and to some extent PB, can be viewed as generalizing pure SAT. The domain of answer set programming (ASP), in contrast, was born in the field of logic programming and comes with a rich history in its own right. The advantages of ASP include an expressive standard modeling language, which allows problems to be specified concisely and declaratively. This domain was described by Section 2.4. The set of competitive solvers aimed at ASP is small, and the clasp system (Gebser et al., 2007) has become dominant; it and its variants took first through third place in the 2011 ASP competition. The suite of constituent solvers will therefore consist entirely of different parameterizations of clasp. Both the new domain and this unusual style of solver suite thus pose an interesting new test of portfolio performance.

An existing portfolio system, claspfolio, uses SVR to select from a predetermined set of clasp configurations (Gebser et al., 2011b). The following experiment will use the same set of potential configurations as claspfolio, listed in Table 8.6. The experiment will compare claspfolio and the probabilistic architecture with both of them trained on the claspfolio training set, a mix of competition and standard benchmark

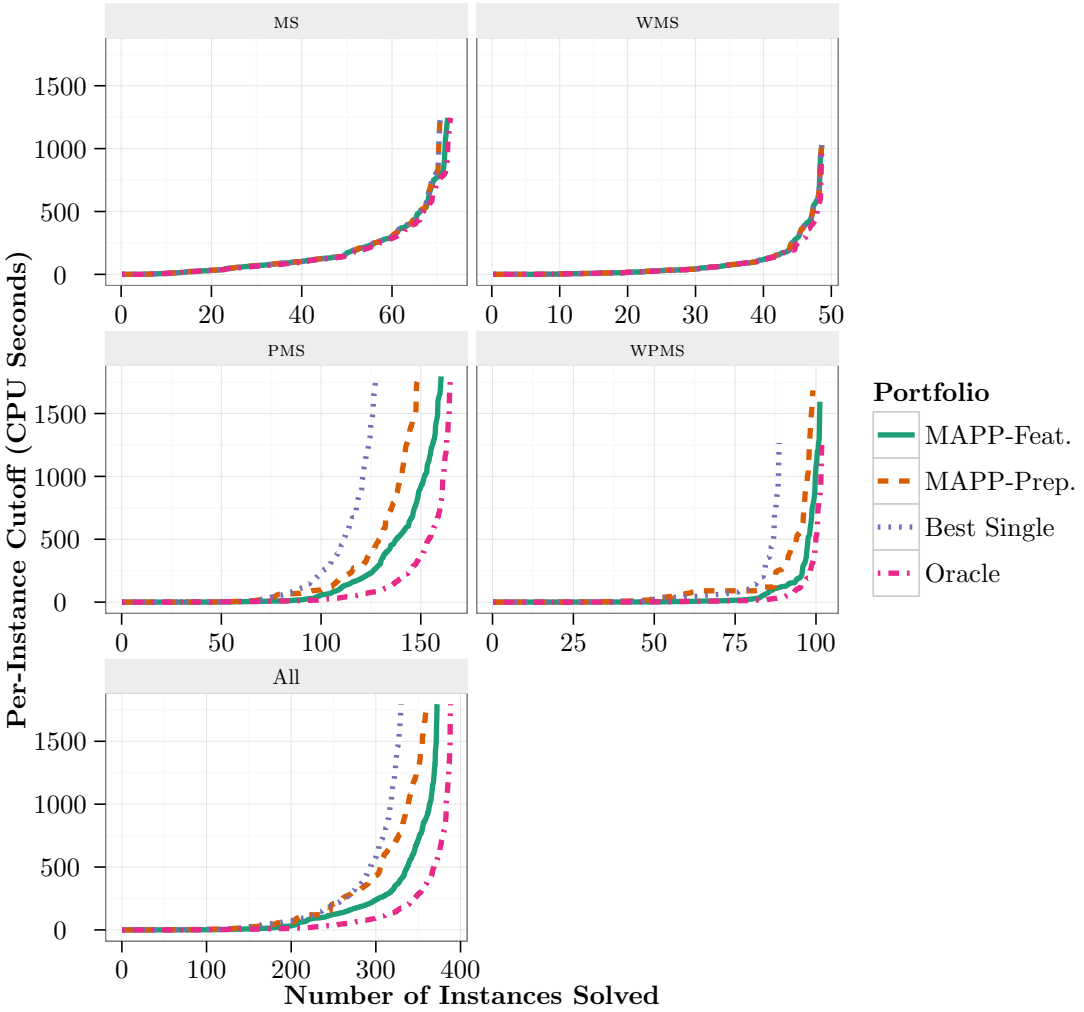


Figure 8.2: Cactus plots comparing the number of MAX-SAT 2011 competition instances solved by reference schemes and MAPP, overall and in each competition category. Substantial differences are visible on the partial MAX-SAT categories (PMS and WPMS), on which the probabilistic portfolio performs best. The advantage offered by MAPP is slight on MS and nonexistent on WMS, but even the oracle portfolio is unable to beat the baseline to any greater degree on these categories.

instances. This training set is unusual in that the majority of its instances are trivial for `clasp`—i.e., all configurations solve them in under one CPU second—and is therefore a new challenge for MAPP. The same set of instance features will be used by both systems.

Table 8.6: Suite of `clasp` configurations used in `claspfolio`, and by the comparison in this section. Even these small differences between parameter settings provide sufficient performance diversity to enable a portfolio to substantially outperform the default configuration. Note that `clasp` is deterministic in all of these configurations. Although MAPP is particularly suited to nondeterministic solvers, it also performs well here.

#	Arguments
1	
2	`--sat-pre=20,25,120 --trans-ext=dynamic --backprop`
3	`--sat-pre=20,25,120`
4	`--eq=20 --backprop`
5	`--restarts=16000`
6	`--restarts=100,1.5,1000`
7	`--restarts=100,1.5,1000 --local-restart`
8	`--restarts=100,1.5,1000 --save-progress`
9	`--restarts=100,1.5,1000 --local-restart --save-progress`
10	`--restarts=256`
11	`--restarts=256 --save-progress`
12	`--restarts=256 --local-restart`
13	`--restarts=256 --local-restart --save-progress`

Table 8.6: Continued.

14	`--heuristic=VSIDS`
15	`--heuristic=Berkmin --berk-max=512`
16	`--heuristic=Berkmin --berk-max=512 --berk-huang`
17	`--heuristic=Berkmin --berk-max=512 --berk-once`
18	`--deletion=1,1,1 --reduce-on-restart`
19	`--dinit=800,10000 --dsched=5000,1.1 --deletion=3,1.1,10000`
20	`--heu=VSIDS --sat-pre=20,25,120 --trans-ext=integ`
21	`--heu=VSIDS --sat-pre=20,25,120 --trans-ext=dynamic`
22	`--sat-pre=20,25,120 --trans-ext=dynamic --initial-look=10 --restarts=no --recursive-str`
23	`--sat-pre=20,25,120 --trans-ext=dynamic --initial-look=10 --restarts=no --recursive-str`
24	`--otfs=1 --recursive-str --reverse-arcs=2 --sat-pre=20,25,120,-1,2`
25	`--heu=VSIDS --sat-pre=20,25,120 --trans-ext=dynamic --loops=no --loops-in-heu=0`

Figure 8.3 presents the outcome of this comparison. All portfolio methods outperform the default `clasp` configuration by wide margins. The baseline best single solver over training instances, listed as configuration 20 in Table 8.6, is much more effective than the default configuration on this instance mixture. Both `claspfolio` and the probabilistic architecture, however, measurably outperform it when they are given feature information. Even though the `clasp` configurations are deterministic, which does not emphasize some of the specific advantages of the probabilistic approach, the performance of MAPP matches

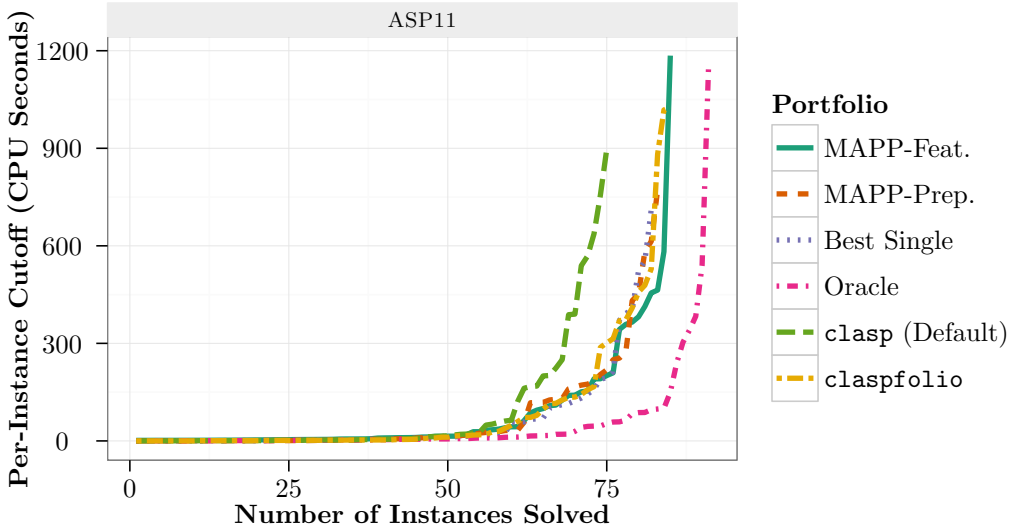


Figure 8.3: Portfolio performance on the instances used in the 2011 ASP competition, with MAPP trained on the instances used to train `claspfolio`. The best single solver and MAPP without features perform almost identically—much better than the default configuration of `clasp`—and are beaten slightly by `claspfolio` and by MAPP with feature information, both of which exhibit very similar performance.

that of the special-purpose portfolio—a portfolio that, on this same set of test instances, won the 2011 ASP competition.

The results of this comparison therefore support the robustness of the probabilistic architecture: in a fourth domain, using an unusual suite of solvers, it again provides a performance advantage over its constituents and remains competitive with the state of the art. These results also identify areas for portfolios to improve in this domain. In particular, the size of remaining gap to oracle performance suggests that additional informative instance features could be valuable.

8.4 Conclusions

The set of experiments in this chapter, spread across three new domains, is instructive, both with respect to the performance of probabilistic portfolios and to the capabilities of portfolios in general. Three lessons in particular may be drawn.

First, taken with the outcomes in SAT, these results reinforce the claim that this architecture is a general-purpose tool. It operated well with small solver suites, such as those in PB and MAX-SAT, with large suites, such as those in SAT, and with an unusual parameterization-based suite in ASP. It was tested over a similar range of instance feature sets. The same underlying decision mechanism was used throughout, and, in all cases, it performed robustly: it pushed performance toward that of oracle when there was room to do so, and tracked baseline performance when there was not.

Second, the inclusion of `claspfolio` in the ASP experiment shows that the architecture compares well with the state of the art performance outside of SAT as well. This result suggests that general-purpose portfolio decision making can be developed and can be kept separate from domain-specific solver and instance interaction.

Third, the engineering effort required to build a portfolio in a new domain is not prohibitively expensive. The feature sets in PB and MAX-SAT are modest, build on existing research in other domains, and include a common set of feature types: most often, measures of instance size, graph properties representing constrainedness, and simple statistics of an instance's raw coefficients. The supported class of solvers, i.e., Las Vegas algorithms, includes solvers for a range of difficult computational problems, including those discussed in this dissertation as well as domains such as SMT and QBF. Competitions in these areas often drive solvers to support a rough standard pattern for user interaction, and for instance and solution representation.

Overall, the experiments in this chapter and in Chapter 7 establish MAPP as broadly effective on multiple domains. The dissertation now shifts from presenting and evaluating MAPP to taking stock of the results obtained. The next chapter considers these lessons

learned, and how they should be applied to future research.

Chapter 9

Discussion and Future Work

The preceding chapters have described MAPP, the details of its components, and its performance in a number of experimental settings. These results are positive, and MAPP should raise expectations for what can be achieved by an algorithm portfolio method. Another important reason to engage in such a research project, however, is to raise new questions and to arrive at new means of making progress toward a larger goal. This chapter therefore considers MAPP critically and in hindsight, with an eye toward applying the lessons learned. It discusses insights to be drawn from its development, both positive and negative, encouragement that can be taken from its results, and potential paths forward in relevant research directions. Organized around the three core MAPP components, it examines their strengths, expands on their weaknesses, describes the most promising options for improving them, and proposes their application beyond the standard algorithm portfolio problem.

9.1 The Modeling Component

Solver performance models are a conceptually vital aspect of MAPP, bridging the gap between the portfolio's observations in training, i.e., sets of solver runs, and its belief state, i.e., sets of run time distributions. Significant statistical machinery, however, is not always

required for the portfolio to perform well overall: given a systematically sampled set of training runs, the simple multinomial model is often sufficient. More complex mixture models can be a valuable tool for gaining insight into solver performance, as discussed in Section 4.7, and they can improve portfolio beliefs in situations where frequent gaps in sampling exist in the training data, as described in Section 7.2. Ideally, however, a single model would handle both roles.

The experience gained in developing and evaluating these models suggests a way forward. Consider two of their primary disadvantages. First, the use of a discrete distribution entails obvious drawbacks, such as the spurious distinction engendered between nearly identical runs that fall across a bin boundary. Second, the general-purpose distributions employed, such as the multinomial, provide significant flexibility at a cost: a model's goal is to infer distributions from limited data, but learning each multinomial distribution involves fitting values to as many parameters as there are performance bins, typically on the order of $B \approx 60$. In practice, data is not so limited as to make this task futile, but is certainly limited enough to motivate a search for alternatives.

Ideally, then, a performance model would represent each run time distribution with fewer parameters. Doing so implies making a stronger statistical assumption about the form of those distributions, and thus implies making a well-justified decision about what distribution is most appropriate. Speculatively, are any candidate distributions available? The "three-parameter" log-normal distribution is one. Widely used in survival analysis and elsewhere (Kalbfleisch and Prentice, 2011), it embodies the assumption that the log of a random variable is normally distributed, or, equivalently, that a random variable is the *product* of many arbitrarily distributed random variables. Its properties seem appropriate for the portfolio setting: it is effectively heavy-tailed, an important and commonly-observed feature of solver behavior (Gomes et al., 2000), and it includes an explicit starting time that corresponds to a solver's minimum reasonable run time. Choosing a distribution correctly for this task, however, requires considering other candidates carefully, such as the Weibull

distribution and truly heavy-tailed "power law" distributions, and an extensive empirical analysis of many solvers' behavior across many instances of multiple domains.

Using a continuous alternative to the multinomial does not require altering the essential model structure developed in this dissertation. The key insight of these models—the value of the mixture structure for inferring RTDs—can be kept intact. Furthermore, this change would not require moving fully away from the discrete nature of these models, since the underlying continuous distribution can be used to parameterize a multinomial distribution deterministically. Discretization at the modeling level, in fact, appears to be a virtue, in that it keeps the modeling and planning components operating on the same representation. Large changes in the density assigned by a continuous distribution may correspond to minimal changes in the probabilities of performance bins and no changes to the derived plans; model discretization mitigates this mismatch.

Such interaction between modeling and planning is another area that deserves further study. The goals of these two components do not perfectly mesh: fitting a model optimizes the probability of data, while planning optimizes the probability of success. An estimated RTD that assigns low probability to run data may, nonetheless, result in plans with a much higher probability of success than those yielded by an estimated RTD that assigns greater probability to run data. Take a simple example involving two estimated RTDs, with three performance bins representing short, long, and failing runs respectively. Let the first distribution place 10% of its probability mass in the first bin, and let the second distribution place none of its mass there. If, in reality, a run falls into the first bin with 1% probability, then the second distribution is an extremely poor model of the data: it assigns zero probability to an event that does, in fact, occur, albeit rarely. In planning, however, the first distribution may lead to plans involving frequent short runs—while the second distribution leads to plans involving the long runs that are truly necessary. This pitfall is important to consider during model development, and is one reason that a distribution capable of placing zero density over low values explicitly, such as the three-parameter log normal, is an appeal-

ing choice. Somehow altering the model-fitting objective to better suit planning, however, might solve this issue with more generality. This dissertation has laid out the framework and demonstrated the promise of solver performance modeling. Future research may tap its full potential.

9.2 The Feature Integration Component

Related to the future modeling work proposed above are research directions involving new sources of, and new uses for, feature information. These directions can be divided into three areas: first, better features, particularly dynamic features; second, the incorporation of feature information into RTD estimation; and, third, incremental improvements to the feature integration scheme presented in Chapter 5.

A more complete solver performance model would incorporate dynamic features of solver runs. Iterative improvements in the value of the objective function in domains such as MAX-SAT and PB, for example, provide a time series that might allow stalled search paths to be detected more quickly. Work along these lines has been undertaken in the area of online algorithm control (Hansen and Zilberstein, 2001; Nareyek, 2004; Cicirello, 2003), but not substantially applied to algorithm portfolios. Similar examples are the search statistics used in the full SATzilla feature set. These statistics are collected during a short, fixed window before true portfolio operation, often by simpler solvers than are used by the portfolio. By instrumenting the solvers that make up the portfolio suite proper, more data could be collected, and new data could be obtained throughout the portfolio's operation on an instance. The greatest obstacle to this improvement is practical. Solver implementations, when they offer any additional information about the status and progress of their search, do so in nonstandard ways that are awkward, at best, to interpret. This reality highlights the need for both solver authors and tool developers, including algorithm portfolio researchers, to develop a common standard for richer interaction with running solvers. Such standards could also aid the development of portfolio methods for important subproblems, such as

incremental SAT.

It may also be useful to integrate models' discriminative and generative components more tightly, or to otherwise use instance features to aid RTD inference—perhaps by sharing statistical strength between superficially similar instances. The current modeling scheme shares statistical strength only among instances whose run data appear similar; no useful performance estimate can be obtained for an instance on which no runs have been observed. Two instances of random k-SAT, however, are more likely, a priori, to yield similar performance distributions. Including feature information in modeling, then, could further improve the sample efficiency of RTD estimation. The goals of RTD estimation and feature modeling, however, may be difficult to balance.

With the pragmatic goal of improving solver performance in mind, incremental improvements to the MAPP feature integration component could yield a more straightforward return on research effort. Although the nearest-M classification objective works well in practice, it is not strictly connected to the true goal of maximizing the probability of a plan's success. A better notion of performance-based distance might involve the success of plans constructed over a pair of instances; under such a definition, two instances are similar if they can be solved by similar plans. Furthermore, identifying a fixed-size set of nearest instances is not the only possible way to cast this problem as classification. One simple alternative would be to label sets of instances on which solvers perform within some threshold of similarity.

The shared goal of both modeling and feature integration is to provide better input to solver scheduling. Potential improvements to that component are discussed next.

9.3 The Scheduling Component

While the pure dynamic programming planner presented in Chapter 6 has been shown to offer good performance in practice, it is not truly optimal. An example may illustrate this point. Consider the RTDs presented in Table 9.1, for which dynamic programming com-

		Success Prob.		
Solver	**Cutoff (s)**	**X**	**Y**	**Z**
A	10	0.2	1.0	0.0
	20	0.2	1.0	0.0
B	10	1.0	0.0	0.0
	20	1.0	0.0	0.0
C	10	0.0	1.0	0.1
	20	0.0	1.0	0.1

Table 9.1: Probabilities of solver success under two different run times in an example situation, constructed to illustrate that the dynamic programming planner, while consistently effective, is not truly optimal. This situation involves three solvers, A, B, and C, and three possible performance scenarios, X, Y, and Z, each equally likely a priori. Dynamic programming first selects a plan for 10 s remaining, choosing to run A. For 20 s remaining, it chooses to run B for 10 s, then its plan for 10 s remaining, i.e., A. The optimal plan, however, is to run C followed by B.

putes the plan $\langle \mathtt{B}, \mathtt{A} \rangle$. That plan succeeds with probability $\frac{2}{3}$. An optimal but exponential-time algorithm instead generates the plan $\langle \mathtt{C}, \mathtt{B} \rangle$, whose probability of success is $\frac{7}{10}$. Because the dynamic programming approach proceeds backwards in time, it suffers from effective amnesia: its choice of final run in this example fails to take advantage of the information gained from an earlier failure. Moving beyond pure dynamic programming might improve its performance. For example, an optimal planner could be employed out to a fixed number of steps, and the dynamic programming approximation used after that point.

It may be more rewarding to allow greater flexibility in the targeted objective, however, than to take the last few, difficult steps toward it. In particular, the current objective focuses exclusively on the probability of success, ignoring efficiency entirely. While maxi-

mizing the chance of success is almost always a substantial component of a user's true goal, it is rarely the only component. Users, even those with significant computational resources to employ, care about how quickly their problems are solved. In many situations, they also care about the quality of those solutions. The planning algorithm should allow the user to trade off multiple goals. How best to specify that tradeoff, however—for example, whether as a linear weighting between goals, as an exponential discount factor, or as the entire set of plans along the Pareto front—is unclear. Progress along these lines may benefit from existing work on similar problems, such as multiobjective optimization, where the user often participates in the optimization process (Miettinen et al., 2008).

It could also be productive to understand more deeply the relationship between solver execution planning, as developed in Chapter 6, and existing variants of the UKP, such as the multiobjective knapsack problem (Kellerer et al., 2004). Along with greater understanding, finding connections to such applications could simplify the derivation of tighter formal bounds on the optimality of planned schedules. Formal bounds could offer insight into future directions for planning.

9.4 Parallelism and Beyond

Potential new applications of portfolio methods are numerous, and MAPP, as a particularly flexible architecture, presents a particularly useful foundation for new work on such applications. Interesting options include parallel portfolios, single-instance portfolios, and sample-efficient algorithm configuration. Each option is discussed below.

Parallel portfolios are an area of ongoing research and of substantial interest (Hamadi, 2009). Existing parallel SAT solvers, such as `ManySAT`, often employ what is effectively a portfolio approach (Hamadi et al., 2009), and the naïve portfolio `ppfolio` was successful in wall-clock comparisons in the 2011 SAT competition (Järvisalo et al., 2011b). Furthermore, the overhead of interaction between constituent search threads in a parallel solver, via mechanisms such as clause sharing, is not always outweighed by its benefits; for this

reason, some successful parallel solvers, such as `plingeling`, share only learned unit clauses (Biere, 2010). Significant interaction between solvers may not, therefore, be necessary. These observations suggest that it may be possible to generalize sequential algorithm portfolio approaches into good parallel methods, and that the advantages of MAPP in the sequential case may also carry over. For example, it may be possible to extend the dynamic programming planner directly to the multicore case. Little else about the architecture would require modification.

Another paradigm for parallel portfolio operation, and especially for cluster-scale computation (Bordeaux et al., 2009), is the division of a single problem instance into multiple independent subproblems. In the simplest such divide-and-conquer approach, the addition of a unit clause to a CNF formula, and its contradiction to a copy of the original formula, yields two versions of the problem under different assumptions. The critical division between satisfiable and unsatisfiable instances applies to these subproblems as well, since a satisfying assignment to a subproblem corresponds to a satisfying assignment to the complete instance, but the vast majority of the subproblems of any nontrivial instance will be unsatisfiable. It would be interesting to evaluate whether different heuristic search strategies exhibit complementary strengths across subproblems of a single instance.

In a different direction, solver performance models provide a general methodology for extracting information from run time data, and could thus play a role in automatic algorithm configuration systems, such as those of Hutter et al. (2009) and Xu et al. (2011). These systems are, to some extent, orthogonal to portfolio methods: they optimize the performance of a single solver for a specific class of instance, and therefore offer the most value when instances are similar. A portfolio method, in contrast, offers the most value when instances are different. Because algorithm configuration methods search through the space of parameterizations by repeatedly collecting run data on a group of instances, a solver performance model could reduce the number of runs required to understand a configuration's overall performance.

Portfolio methods are also orthogonal to systems that apply learning to lower-level solver decisions, as in the polarity prediction work of Silverthorn and Miikkulainen (2011). Such systems assume that instances are similar, and learn how to solve them more quickly by emulating the decisions that led to success in previous runs. Integrating such functionality into a portfolio architecture could yield a system that employs multiple strategies across a heterogeneous group of instances, identifies distinct subgroups, and exploits the predictable aspects of each subgroup.

Finally, simple applications of scheduling and modeling could add value to solver competitions. The preplanning MAPP portfolio performs surprisingly well, despite its ignorance of domain-specific feature information. Such a portfolio makes an excellent baseline method, and would be an effective replacement in the role of competition foil currently played by a hand-selected parallel portfolio such as `ppfolio`. These competitions, in SAT and elsewhere, also generate valuable data on solver performance. That value could be increased by employing a solver performance model to estimate solvers' complete RTDs on competition instances. Occasional repeated runs of a solver on an instance would allow the model to learn how consistently a solver performs.

9.5 Conclusions

Overall, it is instructive to revisit the results of Chapters 7 to 8. The experiments in these chapters demonstrate the present capabilities of portfolio methods, which consistently solve large collections of instances more quickly and more reliably than any individual solver. Even the best methods, however, substantially underperform the oracle limit in every situation—even when, as in Fig. 7.2, competing portfolios choose between only three different solvers. Such results highlight the opportunity that remains for advances in this area, some of which could be realized by the work described this chapter.

It speaks well of the flexibility of MAPP that much of that work can be performed without altering its high-level structure. Regardless of the role that its concrete building

blocks end up playing in future systems, however, MAPP emphasizes conceptual questions that seem likely to remain important. How should the behavior of complex, nondeterministic solvers be inferred from data? How should uncertainty about that behavior be characterized? How can plans be computed under that characterization? What planning objective should be optimized? This dissertation has offered provisional responses to such questions, but not conclusive answers. The final chapter reviews the contributions made by this dissertation, and the progress that MAPP makes toward the larger goals of computer science as a whole.

Chapter 10

Conclusions

This final chapter summarizes the key contributions of this dissertation: MAPP, the components developed to make it possible, and the software toolkit that implements it. It then broadly relates the dissertation to the history and direction of relevant research in AI, and to the field's progress toward the larger goal of efficient, widely applicable, and fully automated logical reasoning.

10.1 Summary of Contributions

The architecture of MAPP is the primary contribution of this dissertation. This architecture centers the operation of an algorithm portfolio around a clear organizing principle: its representation of uncertainty as a finite, weighted set of possible performance scenarios. Each component of the architecture manipulates this representation, and is permitted broad flexibility in how the requirements placed on it are achieved. MAPP offers new capabilities unavailable from other portfolio methods, such as tailored restarts of nondeterministic solvers. It improves on existing portfolio capabilities, as in planning and feature integration. Finally, instantiations of the architecture are successful in four different domains, using a variety of solver suites, benchmark collections, and feature sets, when compared against

a cross-section of the portfolio state of the art. The architecture, then, is both novel and effective. Furthermore, the three main components of MAPP are important contributions in their own right.

The first component applies generative modeling to the estimation of possible performance scenarios from observed run data. This work is valuable in two ways: it introduced solver performance modeling in general, and sample-efficient RTD estimation in particular, as problems worth solving, and proposed and demonstrated the applicability of the mixture-model structure to solving them. In practice, a MAPP portfolio performs well even when a simple multinomial model is used, but more complex models provide good estimates under a larger variety of sampling strategies.

The second component incorporates information about an instance's appearance into a model's estimates of solver performance. This feature information is used to enable the supervised prediction of instance similarity, and the component maintains robustness to poor features by defining similarity in terms of solver performance. This approach builds on the reliability of well-studied classifiers, and works well in combination with solver performance modeling, the MAPP representation of uncertainty, and the component responsible for solver scheduling.

That third component provides a straightforward and efficient solution to the problem of planning solver execution given the information provided by the other pieces of MAPP. By applying dynamic programming to this scheduling problem, it strikes a critical balance between effectiveness and computational complexity: it maintains efficiency in the case of uncertainty by ignoring the information gained from earlier solver runs, but provides optimal planning in the case of perfect information.

Finally, while its presence has remained in the background of this dissertation, enabling the experimental results presented throughout, the custom software used to implement MAPP is an important contribution as well. This `borg` toolkit for algorithm portfolio development provides a straightforward system for collecting solver performance data, as-

sembling solver suites, interacting with domain-specific features, training multiple portfolio methods, and using them to solve problems through a consistent interface. Its flexibility was demonstrated in the construction of portfolios for the variety of domains and solvers explored in this dissertation, and its practicality was demonstrated in competition environments. It is publicly available, easily installed, and a valuable aid both to those conducting research on algorithm portfolio methods and to those tackling problems with modern solvers (Lierler et al., 2012).

10.2 Closing Thoughts

Algorithm portfolios have become powerful tools for solving instances of difficult and important decision and optimization domains. The major contributions of this dissertation are involved in developing the capability, and in demonstrating the benefits, of accounting for the probabilistic behavior of the heuristic algorithms deployed by such tools. This dissertation showed that, by doing so, these tools can be made both more powerful and more general, and the portfolio framework it introduced further amplifies our ability to solve challenging computational problems.

Consider, as well, the role that portfolios may play in the larger scope of research in AI. The ability of heuristic algorithms to solve difficult problems outstrips our understanding of when and how they solve them. The story of automated logical reasoning over the past two decades, the transition from surprising to impressive to frequent success, remains hobbled by this lack of predictability: human experience has taken its place, and the human bottleneck now holds back the spread of automation and the goals of AI. The recent history of solver development should contradict any expectation that these fields are converging on a single implementation or a single algorithm that would obviate the need for judgment based in experience. Instead, we continue to witness new additions to the heuristic zoo, as researchers pursue their separate interests and individual goals.

This situation is unlikely to change, nor should it: it is *because* we can tailor our

algorithms to specific problems, because we can benefit from experience in addition to analysis, that AI escapes from the complexity trap. Rather than frustrate, this situation should further motivate the application of statistical tools, formal methods for learning from experience, to computational problems. The use of a heuristic calls for two responses: first, if possible, its formal understanding and theoretical justification; and, second, a principled methodology for its use. Portfolios can provide this second response. Logical reasoning will be truly automatic only when human input is unnecessary, when problem-solving systems themselves learn from the experience of solving problems—as humans do, but in ways that humans cannot. The algorithm portfolio paradigm is a practical vehicle for moving toward that goal. This dissertation moves it farther down the road.

Bibliography

Arbelaez, A., Hamadi, Y., and Sebag, M. (2010). Building portfolios for the protein structure prediction problem. In *Workshop on Constraint Based Methods for Bioinformatics*.

Audemard, G., Katsirelos, G., and Simon, L. (2010). A restriction of extended resolution for clause learning SAT solvers. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*.

Auer, P., Bianchi, N. C., Freund, Y., and Schapire, R. E. (1995). Gambling in a rigged casino : the adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*.

Auer, P., Bianchi, N. C., Freund, Y., and Schapire, R. E. (2002a). The nonstochastic multi-armed bandit problem. *SIAM Journal on Computing*.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002b). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*.

Babai, L. (1979). Monte-Carlo algorithms in graph isomorphism testing. Technical report, Université de Montréal.

Balint, A., Gall, D., Kapler, G., and Retz, R. (2010). Experiment design and administration for computer clusters for SAT-solvers (EDACC). *Journal on Satisability, Boolean Modeling and Computation (JSAT)*.

Baral, C. (2003). *Knowledge representation, reasoning and declarative problem solving*.

Batory, D. (2005). Feature models, grammars, and propositional formulas. In *Software Product Lines*.

Bellman, R. (1957). *Dynamic programming*.

Biere, A. (2010). Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. Technical report, Institute for Formal Models and Verification, Johannes Kepler University.

Biere, A., Cimatti, A., Clarke, E. M., Fujita, M., and Zhu, Y. (1999). Symbolic model checking using SAT procedures instead of BDDs. In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference (DAC)*.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research*.

Blum, A., and Mansour, Y. (2007). From external to internal regret. *Journal of Machine Learning Research (JMLR)*.

Bordeaux, L., Hamadi, Y., and Samulowitz, H. (2009). Experiments with massively parallel constraint solving. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*.

Brélaz, D. (1979). New methods to color the vertices of a graph. *Communications of the ACM (CACM)*.

Brooks, D. R., Erdem, E., Erdoğan, S. T., Minett, J. W., and Ringe, D. (2007). Inferring phylogenetic trees using answer set programming. *Journal of Automated Reasoning*.

Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Hanna, Z., Nadel, A., Palti, A., and Sebastiani, R. (2007). A lazy and layered SMT(BV) solver for hard industrial verification problems. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV)*.

Cesa-Bianchi, N., Lugosi, G., and Stoltz, G. (2004). Minimizing regret with label efficient prediction. *Proceedings of the 17th Annual Conference on Learning Theory (COLT)*.

Cicirello, V., and Smith, S. (2005). The max k-armed bandit : a new model for exploration applied to search heuristic selection. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*.

Cicirello, V. A. (2003). *Boosting stochastic problem solvers through online self-analysis of performance*. PhD thesis.

Clarke, E., Biere, A., Raimi, R., and Zhu, Y. (2001). Bounded model checking using satisfiability solving. *Formal Methods in System Design*.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)*.

Crawford, J. M., and Auton, L. D. (1996). Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM (CACM)*.

de Moura, L., and Bjørner, N. (2010). Bugs, moles and skeletons : symbolic reasoning for software development. In *International Joint Conference on Automated Reasoning (IJCAR)*.

de Moura, L., and Bjørner, N. (2011). Applications and challenges in satisfiability modulo theories. In *Workshop on Invariant Generation (WING)*.

Dequen, G., and Dubois, O. (2004). `kcnfs` : an efficient solver for random k-SAT formulae. In *Theory and Applications of Satisfiability Testing (SAT)*.

Dixon, H. E., and Ginsberg, M. L. (2000). Combining satisfiability techniques from AI and OR. *Knowledge Engineering Review*.

Doyle, G., and Elkan, C. (2009). Accounting for word burstiness in topic models. In *Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML)*.

Eén, N. (2011). Introduction to satisfiability solving with practical applications. Presentation at the SAT/SMT Summer School.

Eén, N., and Sörensson, N. (2004). An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*.

Eén, N., and Sörensson, N. (2006). Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*.

Elkan, C. (2006). Clustering documents with an exponential-family approximation of the Dirichlet compound multinomial distribution. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML)*.

Erkök, L., Carlsson, M., and Wick, A. (2009). Hardware/software co-verification of cryptographic algorithms using cryptol. In *Formal Methods in Computer-Aided Design (FMCAD)*.

Fan, R. E., Chang, K. W., Hsieh, C. J., Wang, X. R., and Lin, C. J. (2008). LIBLINEAR : a library for large linear classification. *Journal of Machine Learning Research*.

Fei-Fei, L., and Perona, P. (2005). A Bayesian hierarchical model for learning natural scene categories. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.

Feige, U., Lovász, L., and Tetali, P. (2004). Approximating min sum set cover. *Algorithmica*.

Gagliolo, M., and Schmidhuber, J. (2006). Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*.

Gagliolo, M., and Schmidhuber, J. (2007). Learning restart strategies. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*.

Gagliolo, M., and Schmidhuber, J. (2011). Algorithm portfolio selection as a bandit problem with unbounded losses. *Annals of Mathematics and Artificial Intelligence*.

Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2011a). Challenges in answer set solving. In *Proceedings of Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning (LPNMR)*.

Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneider, M., and Ziller, S. (2011b). A portfolio solver for answer set programming : preliminary report. In *Logic Programming and Nonmonotonic Reasoning*.

Gebser, M., Kaufmann, B., Neumann, A., and Schaub, T. (2007). Conflict-driven answer set solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*.

Gelfond, M., and Leone, N. (2002). Logic programming and knowledge representation—the A-Prolog perspective. *Artificial Intelligence*.

Gerrish, S. M., and Blei, D. M. (2010). A language-based approach to measuring scholarly impact. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*.

Giunchiglia, E., Lierler, Y., and Maratea, M. (2006). Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning*.

Gogate, V., and Domingos, P. (2011). Probabilistic theorem proving. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*.

Gomes, C. P., Kautz, H., Sabharwal, A., and Selman, B. (2008). *Satisfiability solvers*.

Gomes, C. P., and Selman, B. (1997a). Algorithm portfolio design : theory vs. practice. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI '97)*.

Gomes, C. P., and Selman, B. (1997b). Problem structure in the presence of perturbations. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI)*.

Gomes, C. P., Selman, B., Crato, N., and Kautz, H. (2000). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*.

Gomes, C. P., Selman, B., and Kautz, H. (1998). Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI)*.

Hahsler, M., Hornik, K., and Buchta, C. (2008). Getting things in order : an introduction to the R package seriation. *Journal of Statistical Software*.

Hamadi, Y. (2009). From SAT to parallel SAT solving. Tutorial at Learning and Intelligent Optimization (LION).

Hamadi, Y., Jabbour, S., and Sais, L. (2009). ManySAT : a parallel SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*.

Hansen, E. A., and Zilberstein, S. (2001). Monitoring and control of anytime algorithms : a dynamic programming approach. *Artificial Intelligence*.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning*.

Horvitz, E., Ruan, Y., Gomes, C. P., Kautz, H. A., Selman, B., and Chickering, D. M. (2001). A Bayesian approach to tackling hard computational problems. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI)*.

Houstis, E. N., Catlin, A. C., Rice, J. R., Verykios, V. S., Ramakrishnan, N., and Houstis, C. E. (2000). PYTHIA-II : a knowledge/database system for managing performance data and recommending scientific software. *ACM Transactions on Mathematical Software*.

Huang, J. (2007). The effect of restarts on the efficiency of clause learning. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI)*.

Huberman, B., Lukose, R., and Hogg, T. (1997). An economics approach to hard computational problems. *Science*.

Hutter, F., Babić, D., Hoos, H. H., and Hu, A. J. (2007). Boosting verification by automatic tuning of decision procedures. In *Proceedings of Formal Methods in Computer Aided Design (FMCAD)*.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009). ParamILS : an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research (JAIR)*.

Järvisalo, M., Le Berre, D., and Roussel, O. (2011a). The SAT 2011 competition results : part 2. Presentation at Theory and Applications of Satisfiability Testing (SAT).

Järvisalo, M., Le Berre, D., and Roussel, O. (2011b). SAT competition results. `http://www.satcompetition.org/2011/`.

Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., and Sellmann, M. (2011). Algorithm selection and scheduling. In *17th International Conference on Principles and Practice of Constraint Programming (CP)*.

Kadioglu, S., Malitsky, Y., Sellmann, M., and Tierney, K. (2010). ISAC—instance-specific algorithm configuration. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI)*.

Kaivola, R., Ghughal, R., Narasimhan, N., Telfer, A., Whittemore, J., Pandav, S., Slobodová, A., Taylor, C., Frolov, V., Reeber, E., and Naik, A. (2009). Replacing testing with formal verification in Intel Core i7 processor execution engine validation. In *Computer Aided Verification (CAV)*.

Kalbfleisch, J. D., and Prentice, R. L. (2011). *The statistical analysis of failure time data*.

Karp, R. M. (2011). Heuristic algorithms in computational molecular biology. *Journal of Computer and System Sciences*.

Katz, S. M. (1996). Distribution of content words and phrases in text and language modelling. *Natural Language Engineering*.

Kautz, H., Horvitz, E., Ruan, Y., Gomes, C., and Selman, B. (2002). Dynamic restart policies. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*.

Kautz, H., and Selman, B. (1992). Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI)*.

Kautz, H., and Selman, B. (2007). The state of SAT. *Discrete Applied Mathematics*.

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack problems*.

Koller, D., and Friedman, N. (2009). *Probabilistic graphical models*.

Kroer, C., and Malitsky, Y. (2011). Feature filtering for instance-specific algorithm configuration. In *23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*.

Lai, T. L. (1987). Adaptive treatment allocation and the multi-armed bandit problem. *The Annals of Statistics*.

Le Berre, D. (2011). A brief introduction to practical SAT solving. Presentation at the Tübingen SAT Workshop.

Le Berre, D., and Rapicault, P. (2009). Dependency management for the Eclipse ecosystem : Eclipse p2, metadata and resolution. In *Proceedings of the 1st International Workshop on Open Component Ecosystems (IWOCE)*.

Le Berre, D., Roussel, O., and Simon, L. (2007). SAT competition results. http://www.satcompetition.org/2007/.

Le Berre, D., Roussel, O., and Simon, L. (2009). SAT competition results. http://www.satcompetition.org/2009/.

Leyton-Brown, K., Nudelman, E., and Shoham, Y. (2002). Learning the empirical hardness of optimization problems : the case of combinatorial auctions. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP)*.

Li, L., Zhou, M., Sapiro, G., and Carin, L. (2011). On the integration of topic modeling and dictionary learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*.

Lierler, Y., and Maratea, M. (2004). Cmodels-2 : SAT-based answer set solver enhanced to non-tight programs. In *Proceedings of Logic Programming and Nonmonotonic Reasoning (LPNMR)*.

Lierler, Y., Silverthorn, B., and Schneider, M. (2012). Surviving solver sensitivity : an ASP practitioner's guide. Under review.

Lifschitz, V. (2008). What is answer set programming? In *Proceedings of the 23rd National Conference on Artificial Intelligence*.

Luby, M., Sinclair, A., and Zuckerman, D. (1993). Optimal speedup of Las Vegas algorithms. *Information Processing Letters*.

Madsen, R. E., Kauchak, D., and Elkan, C. (2005). Modeling word burstiness using the Dirichlet distribution. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML)*.

Malitsky, Y., Sabharwal, A., Samulowitz, H., and Sellmann, M. (2011). Non-model-based algorithm portfolios for SAT. In *Theory and Applications of Satisfiability Testing (SAT)*.

Manquinho, V., and Roussel, O. (2010). PB competition results. `http://www.cril.univ-artois.fr/PB10/`.

Manquinho, V., and Roussel, O. (2011). PB competition results. `http://www.cril.univ-artois.fr/PB11/`.

Markowitz, H. (1952). Portfolio selection. *Journal of Finance*.

Marques-Silva, J. a. P., and Sakallah, K. A. (1999). GRASP : a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*.

Matos, P., Planes, J., Letombe, F., and Marques-Silva, J. a. (2008). A MAX-SAT algorithm portfolio. In *Proceedings of 18th European Conference on Artificial Intelligence (ECAI)*.

McCallum, A., Wang, X., and Mohanty, N. (2007). Joint group and topic discovery from relations and text. In *Statistical Network Analysis: Models, Issues, and New Directions*.

Messelis, T., and De Causmaecker, P. (2010). An NRP feature set. Technical report, Katholieke Universiteit Leuven.

Messelis, T., and De Causmaecker, P. (2011). An algorithm selection approach for nurse rostering. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence*.

Miettinen, K., Ruiz, F., and Wierzbicki, A. P. (2008). Introduction to multiobjective optimization : interactive approaches. In *Multiobjective Optimization*.

Mimno, D. (2009). Reconstructing Pompeian households. In *Applications of Topic Models Workshop at Neural Information Processing Systems (NIPS)*.

Mimno, D., Wallach, H. M., Naradowsky, J., Smith, D. A., and McCallum, A. (2009). Polylingual topic models. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Minka, T. P. (2009). Estimating a Dirichlet distribution. Technical report.

Mitchell, D., Selman, B., and Levesque, H. J. (1996). Generating hard satisfiability problems. *Artificial Intelligence*.

Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., and Troyansky, L. (1999). Determining computational complexity from characteristic 'phase transitions'. *Nature*.

Nadel, A. (2011). SAT technology @ Intel. Presentation at the SAT 2011 Workshop on Pragmatics of SAT.

Nareyek, A. (2004). *Choosing search heuristics by non-stationary reinforcement learning*.

Naveh, Y. (2010). The big deal : applying constraint satisfaction technologies where it makes the difference. In *Theory and Applications of Satisfiability Testing (SAT)*.

Nieuwenhui, R. (2009). Professional football scheduling with Barcelogic.

Nikolić, M., Marić, F., and Janičić, P. (2009). Instance-based selection of policies for SAT solvers. In *Theory and Applications of Satisfiability Testing (SAT 2009)*.

Nikolić, M., Marić, F., and Janičić, P. (2011). Simple algorithm portfolio for SAT. *Artificial Intelligence Review*.

Nogueira, M., Balduccini, M., Gelfond, M., Watson, R., and Barry, M. (2001). An A-Prolog decision support system for the space shuttle. In *Practical Aspects of Declarative Languages*.

Nudelman, E., Leyton-Brown, K., Hoos, H. H., Devkar, A., and Shoham, Y. (2004). Understanding random SAT : beyond the clauses-to-variables ratio. In *Principles and Practice of Constraint Programming (CP)*.

O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., and O'Sullivan, B. (2008). Using case-based reasoning in an algorithm portfolio for constraint solving. In *19th Irish Conference on AI*.

Park, J. D. (2002). Using weighted MAX-SAT engines to solve MPE. In *Eighteenth National Conference on Artificial Intelligence*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and E., D. (2011). Scikit-learn : machine learning in Python. *Journal of Machine Learning Research*.

Pulina, L., and Tacchella, A. (2007). A multi-engine solver for quantified Boolean formulas. In *Principles and Practice of Constraint Programming (CP)*.

Pulina, L., and Tacchella, A. (2009). A self-adaptive multi-engine solver for quantified Boolean formulas. *Constraints*.

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*.

Rice, J. R. (1979). Methodology for the algorithm selection problem. In *Proceedings of the IFIP TC 2.5 Working Conference on Performance Evaluation of Numerical Software*.

Roussel, O. (2011). Description of ppfolio. In *SAT Competition 2011*.

Ruan, Y., Horvitz, E., and Kautz, H. (2002). Restart policies with dependence among runs : a dynamic programming approach. In *Principles and Practice of Constraint Programming (CP)*.

Selman, B., Kautz, H. A., and Cohen, B. (1993). Local search strategies for satisfiability testing. In *Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability*.

Selman, B., Kautz, H. A., and Cohen, B. (1994). Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*.

Selman, B., Levesque, H., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI)*.

Silverthorn, B. (2012). Supporting source code and data. `http://nn.cs.utexas.edu/pages/research/borg/`.

Silverthorn, B., and Miikkulainen, R. (2010). Latent class models for algorithm portfolio methods. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Silverthorn, B., and Miikkulainen, R. (2011). Learning polarity from structure in SAT. In *Theory and Applications of Satisfiability Testing (SAT)*.

Sinz, C., and Iser, M. (2009). Problem-sensitive restart heuristics for the DPLL procedure. In *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT)*.

Smith, B. M. (1999). The Brélaz heuristic and optimal static orderings. In *Principles and Practice of Constraint Programming (CP)*.

Soos, M., Nohl, K., and Castelluccia, C. (2009). Extending SAT solvers to cryptographic problems. In *Theory and Applications of Satisfiability Testing (SAT)*.

Streeter, M., Golovin, D., and Smith, S. F. (2007a). Combining multiple heuristics online. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI)*.

Streeter, M., Golovin, D., and Smith, S. F. (2007b). Restart schedules for ensembles of problem instances. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI)*.

Streeter, M., and Smith, S. F. (2007). Using decision procedures efficiently for optimization.

In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS)*.

Streeter, M., and Smith, S. F. (2008). New techniques for algorithm portfolio design. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI)*.

Streeter, M. J., and Smith, S. F. (2006a). An asymptotically optimal algorithm for the max k-armed bandit problem. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)*.

Streeter, M. J., and Smith, S. F. (2006b). A simple distribution-free approach to the max k-armed bandit problem. In *Principles and Practice of Constraint Programming (CP)*.

Tompkins, D. A. D., Balint, A., and Hoos, H. H. (2011). Captain Jack : new variable selection heuristics in local search for SAT. In *Theory and Applications of Satisfiability Testing (SAT)*.

Van Gelder, A., Le Berre, D., Biere, A., Kullmann, O., and Simon, L. (2005). Purse-based scoring for comparison of exponential-time programs. Poster at Theory and Applications of Satisfiability Testing (SAT).

Veneris, A. (2003). Fault diagnosis and logic debugging using boolean satisfiability. In *Proceedings of the 4th International Workshop on Microprocessor Test and Verification: Common Challenges and Solutions*.

Wallach, H. M. (2008). *Structured topic models for language*. PhD thesis, University of Cambridge.

Weerawarana, S., Houstis, E. N., Rice, J. R., Joshi, A., and Houstis, C. E. (1996). PYTHIA : a knowledge-based system to select scientific algorithms. *ACM Transactions on Mathematical Software*.

Wichert, L., and Wilke, R. A. (2005). Application of a simple nonparametric conditional quantile function estimator in unemployment duration analysis. *Social Science Research Network Working Paper Series*.

Wood, R. G., and Rutenbar, R. A. (1997). FPGA routing and routability estimation via boolean satisfiability. In *Proceedings of the Fifth International Symposium on Field-Programmable Gate Arrays (FPGA)*.

Xu, L., Hoos, H., and Leyton-Brown, K. (2007). Hierarchical hardness models for SAT. In *Principles and Practice of Constraint Programming (CP)*.

Xu, L., Hutter, F., Hoos, H., and Leyton-Brown, K. (2009). SATzilla2009: an automatic algorithm portfolio for SAT. In *SAT Competition 2009*.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008a). SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research (JAIR)*.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008b). SATzilla2008 : an automatic algorithm portfolio for SAT. System Description for SAT Race 2008.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Hydra-MIP : automated algorithm configuration and selection for mixed integer programming. In *Proceedings of the 18th RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*.

Vita

Bryan was born in Phoenix, Arizona, and lived there until attending Cornell University in Ithaca, New York, from which he graduated in 2005. After surviving four winters, he chose to move somewhere with less snow, more sun, and better music, entering The University of Texas at Austin in the fall of 2005.

Permanent Address: USA

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay and James A. Bednar.